Henry Joutsijoki (toim.)

Tietojenkäsittelytieteellisiä tutkielmia
Syksy 2018

TAMPEREEN YLIOPISTO

INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 70/2018

TAMPERE 2018

TAMPEREEN YLIOPISTO
INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 70/2018
JOULUKUU 2018

Henry Joutsijoki (toim.)

Tietojenkäsittelytieteellisiä tutkielmia Syksy 2018

ISBN 978-952-03-0934-3 (pdf)

ISSN-L 1799-8158
ISSN 1799-8158

Sisällysluettelo

Mobiilin lisätyn todellisuuden mahdollisuudet ja käytettävyys

Ari Hakala

Tiivistelmä.
Tämän kirjallisuuskatsauksen tavoitteena on selvittää mobiilin lisätyn todellisuuden sovellusten mahdollisuudet ja yleisimmät käytettävyysongelmat. Keskityn ainoastaan niihin sovelluksiin, joissa oppiminen tai turismi on sovelluksen aihealueena. Kirjallisuuskatsauksen aineistona on kuusi tutkimusjulkaisua. Tyypillinen osallistujamäärä oli 20. Aineiston tutkimukset jakaantuvat tasaisesti kvalitatiivisiin, kvantitatiivisiin tai molempia tutkimussuuntia hyödyntäviin tutkimuksiin. Aineiston tutkimuksissa viitattiin suunnitteluohjeisiin ja heuristiikkoihin, mutta vakiintunutta suunnittelukäytäntöä ei ollut. Sovelsin tässä kirjallisuuskatsauksessa yhtä heuristiikkakokoelmaa tunnistamaan käytettävyysongelmia aineiston sovelluksista. Mobiilin lisätyn todellisuuden suunnittelu tulisi johdonmukaistaa. Tällä hetkellä ei ole vakiintunutta tapaa arvioida mobiilin lisätyn todellisuuden erityispiirteiden käytettävyyttä.

Avainsanat ja -sanonnat: mobiili lisätty todellisuus, käytettävyys, oppiminen, turismi.

1. Johdanto

Lisätty todellisuus (augmented reality, AR) on tekniikka, jossa lisätään todellisuuteen lisäkerroksia teknologian avulla. Näin pystytään yhdistämään fyysinen ja virtuaalinen maailma yhdeksi [Höllerer and Feiner 2004]. Yhdistäminen tehdään kuitenkin niin, että fyysinen maailma säilyy aina taustalla. Lisättyä todellisuutta on alettu käyttää yhä enemmän mobiilisovelluksissa. Tätä ilmiötä on alettu kutsua mobiiliksi lisätyksi todellisuudeksi (mobile augmented reality). Mobiilin lisätyn todellisuuden etuja ovat, että sitä pystytään käyttämään missä ja milloin tahansa. [Höllerer and Feiner 2004]

Mobiilin lisätyn todellisuuden sovellukset ovat myös menestyneet markkinoilla. Virtuaalitodellisuutta (virtual reality, VR) ja AR:ä tarkastelevan raportin mukaan VR/AR markkinat saavuttavat 108 miljardin dollarin arvon vuoteen 2021 mennessä [Digi-Capital 2018]. Esimerkkinä tästä voidaan mainita Pokémon Go -peli, joka nousi ilmiöksi. Esittelen sen esimerkkitapauksena tässä kirjallisuuskatsauksessa.

Voidaan siis sanoa, että mobiilin lisätyn todellisuuden markkinoilla on paljon potentiaalia ja sen uutuuden viehätys vetoaa erityisesti nuoriin, jotka ovat Pokémon Go -pelin suurin käyttäjäryhmä [Mansikkamäki 2016]. Pokémon Go on viihdekäyttöön suunniteltu peli, mutta voisiko mobiilia lisättyä todellisuutta

hyödyntää muunlaisessa käytössä? Näen erityisesti potentiaalia turismi- ja oppimissovelluksissa. Mobiili lisätty todellisuus voisi elävöittää oppimista ja se auttaisi turisteja oppimaan esimerkiksi alueen historiasta tai löytämään kahviloita. Tästä johtuen keskityn tässä kirjallisuuskatsauksessa tarkistelemaan sovelluksia, joiden aihealueena on oppiminen tai turismi.

Aikaisemmin tästä aiheesta tehtyjä tutkimuksia ovat tutkineet Väänänen-Vainio-Mattila ja muut [2015] sekä Räty [2017] kirjallisuuskatsauksissaan. Kirjallisuuskatsaukseni eroaa heidän töistään siltä osin, että rajaan sovellusten aihealueeksi oppimisen ja turismin. Lisäksi keskityn myös tarkastelemaan käytettävyysongelmia, joita mobiilin lisätyn todellisuuden sovelluksissa voi ilmetä.

Kirjallisuuskatsaukseni tavoitteena on valita kriteerit täyttävä aineisto ja esitellä sen sisältämien tutkimusten sovelluksen aihealue, tutkimusmenetelmät ja tulokset. Lisäksi tarkastelen ja arvioin käytettävyysongelmia, joita pystyn tunnistamaan sovelluksista. En tutki virtuaalisen todellisuuden sovelluksia. Tämä johtuu siitä, että ne eivät ole vielä tavallisen kuluttajan saavutettavissa. Esteitä ovat esimerkiksi VR-lasien hinta, vaikea saatavuus ja suppea sovellustarjonta. Lisäksi haluan tutkia, miten lisätyn todellisuuden sovelluksia käytetään liikkuessa ja miten se vaikuttaa niiden käyttöön. Tämä rajaus sulkee jo itsessään VR-sovellukset ulkopuolelle, koska niiden käyttö on hyvin paikkaan sidottua.

Tutkielman luvussa 2 esittelen keskeiset käsitteet kirjallisuuteen perustuen. Luvussa 3 esittelen kirjallisuuskatsauksen aineiston: kunkin tutkimuksen sovelluksen aihealueen, tutkimuksen menetelmät ja tutkimuksen tulokset. Lisäksi esittelen esimerkkinä yhden tutkimuksen ja siihen liittyvän keskeisen käytettävyysongelman. Luvussa 4 tunnistan käytettävyysongelmia aineiston sovelluksista ja teen yhteenvedon.

2. Keskeisimmät käsitteet

Seuraavaksi esittelen tutkielman kannalta keskeiset käsitteet: mobiilin lisätyn todellisuuden käsite ja paikannus. Lopuksi tarkastelen käytettävyyttä erityisesti lisätyn todellisuuden mobiilisovelluksissa.

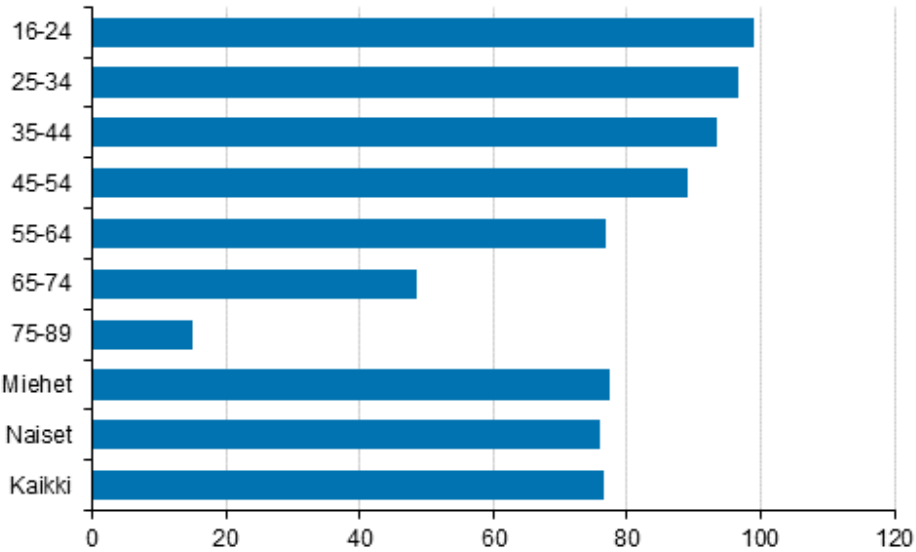
2.1. Mobiili lisätty todellisuus

Lisätyn todellisuuden vastapari on virtuaalitodellisuus ja ne molemmat kuuluvat laajempaan käsitteeseen tehostettu todellisuus (mixed reality) [Milgram and Kishino 1994]. Virtuaalitodellisuus pyrkii luomaan kokonaan uuden digitaalisen ympäristön, jonka kanssa voi vuorovaikuttaa. Lisätty todellisuus pyrkii täydentämään fyysistä maailmaa luomalla sen päälle lisäkerroksia, joissa on

digitaalista sisältöä. Käyttäjä voi siis aina nähdä fyysisen maailman lisätyn todellisuuden kerrosten alla.

Lisätyn todellisuuden kerroksia myös päivitetään reaaliaikaisesti, joten jos kuvakulma vaihtuu niin objektit myös päivittyvät. Lisätyn todellisuuden objektit voivat olla minkälaista digitaalista sisältöä tahansa. Ne voivat olla interaktiivista pelisisältöä tai vain mainoksia tai tietoiskuja. Reaaliaikainen päivittyminen mahdollistaa käyttäjän liikkumisen ja tämä on todella tärkeä ominaisuus lisättyä todellisuutta käyttävissä mobiilisovelluksissa. Reaaliaikaisen päivittymisen ehto on, että laitteen on pystyttävä tunnistamaan käyttäjän laitteen sijainti ja orientaatio. Tätä kutsutaan kontekstitietoisuudeksi [Räty 2017].

Mobiilin lisätyn todellisuuden sovellusten potentiaalisiksi käyttäjiksi voisi kutsua kaikkia älypuhelimen omistavia ihmisiä, mikä Suomen tapauksessa tarkoittaisi noin 80% yli 15-vuotiaista Suomen kansalaisista (kuva 1) [Tilastokeskus 2017]. Mobiilin lisätyn todellisuuden sovellusten etuja ovat hyvä saatavuus ja matala aloituskynnys. Lisättyä todellisuutta hyödyntävät mobiilisovellukset toimivat kaikissa moderneissa älypuhelimissa. Jopa markkinoiden halvimmat älypuhelimet ovat tarpeeksi tehokkaita lisättyä todellisuutta hyödyntäviä mobiilisovelluksia varten.



Kuva 1. Älypuhelin omassa käytössä 2017, %-osuus väestöstä. [Tilastokeskus 2017]

Digitaalinen viihde ja hyötysisältö on fyysisesti haitallista, sillä olemme käyttäneet ja kuluttaneet digitaalista sisältöä pääsääntöisesti päätelaitteen ääressä istumalla. Tämä on ollut fyysiselle kunnolle ja terveydelle haitallista, mutta nyt mobiilin lisätyn todellisuuden avulla lisätty todellisuus on kaikkialla ja kaikille.

Tätä kutsutaan kaikkialla läsnäolevan lisätyn todellisuuden ilmiöksi [Höllerer and Feiner 2004]

Esimerkkitapaus Pokémon Go

Monet saivat ensimmäisen kunnollisen mobiilin lisätyn todellisuuden kokemuksen Pokémon Go :sta. Pokémon Go on vuonna 2016 julkaistu mobiilipeli, joka saavutti yli 500 miljoonaa latauskertaa ilmestymisvuotensa aika [Mansikkamäki 2016], tehden siitä todella suositun mobiilipelin. Se oli myös itselläni ensimmäinen kosketus mobiiliin lisättyyn todellisuuteen, ja se jätti pysyvän vaikutuksen.

Pokémon Go on mobiililaitteella pelattava Pokémon-peli, jonka liikkumiseen tarvittava syöte on käyttäjän itsensä liikkuminen reaalimaailmassa. Riippuen paikasta peliin voi ilmestyä sisältöä kuten Pokémon-hahmo (kuva 2) tai Pokémon stop [Pokémon Go 2017]. Tämä paikka, johon sisältöä voi ilmestyä, on usein joku historiallinen tai muuten erityinen kohde julkisella paikalla. Käyttäjät voivat itse ehdottaa näitä paikkoja, ja tällä tavoin suurin osa näistä Pokémon stop -paikoista on syntynytkin. Pokémon-hahmot ilmestyvät useimmiten näille Pokémon stop -paikoille. Pokémon-hahmo napataan hakemalla se puhelimen näytön keskelle fyysisesti liikuttamalla puhelinta. Tämän jälkeen tehdään heittoele puhelimen kosketusnäytöllä. Tätä tehdessä käyttäjän tulee ottaa huomioon Pokémon-hahmon liikkuminen näytöllä.

Pokémon Go aiheutti suorastaan ”ilmiön” ja se noteerattiinkin laajasti maailmalla. Alussa erityisesti sen vaikutusta nuorten liikkumiseen pidettiin hyvänä asiana [Mansikkamäki 2016], mutta Blaszkiewiczin ja muiden [2017] tutkimus osoitti, että käyttäjän päivittäin kulkema matka ei noussutkaan pelaamisen myötä. Ainoa merkittävä muutos, joka huomattiin tutkimuksessa, oli puhelimen käytön lisääntyminen. Blaszkiewiczin ja muiden tutkimuksessa oli iso otanta ja asioita tarkasteltiin keskiarvoilla, joten isot poikkeamat keskiarvoista ovat mahdollisia. Itse uskon, että jotkut pelaajista kävelivät itsensä uuvuksiin pelatessaan Pokémon Go -peliä.

Pokémon Go -peli suunniteltiin jo alusta alkaen sosiaaliseksi peliksi. Pelaajat saavat alusta asti lisäarvoa pelatessaan yhdessä muiden kanssa. Esimerkiksi Pokémon-hahmoja ilmestyy useammin, jos muita pelaajia on lähellä [Sofian 2017]. Muita sosiaalisuutta lisääviä toimintoja ovat ystävälista, tiimit ja salit, jotka edesauttavat entisestään ihmisiä toimimaan yhdessä.

Kuva 2. Pokémon Go -peliä pelatakseen on harrastettava myös liikuntaa [Smith 2016].

Pokémon Go -pelin saama myönteinen huomio on edesauttanut mobiilia lisättyä todellisuutta hyödyntävien sovellusten markkinoita. Nykyään se on kasvavin mobiilisovellusmarkkina-alue [Statista 2018].

2.2. Paikannus

Kuten jo edellä mainitsin, kontekstitietoisuus on avainasemassa tarkasteltaessa mobiilin lisätyn todellisuuden sovelluksia. Kontekstitietoisuus tarkoittaa sitä, että laite pystyy käsittämään sijaintinsa ja orientaationsa [Räty 2017]. Orientaation puhelimet havaitsevat kiihdytinanturin (accelerator) ja gyroskoopin avulla. Kiihdytinanturi on kaikissa älypuhelimissa perusominaisuutena. Se mahdollistaa näytön kuvan kääntämisen laitteen mukana, mutta se ei pelkästään riitä mobiilin lisätyn todellisuuden sovelluksissa. Esimerkiksi Pokémon Go:n lisätty todellisuus ei toimi pelkästään tällä, vaan tarvitaan tarkempaa tietoa puhelimen asennosta (rotation) ihmisen kädessä. Gyroskooppiteknologian avulla saadaan paljon tarkempaa tietoa puhelimen liikkeistä ja asennosta. Tämä teknologia mahdollistaa lisätyn todellisuuden käytön Pokémon Go -pelissä ja muissa lisättyä todellisuutta hyödyntävissä mobiilisovelluksissa.

Sijaintitietojen käyttö on tärkeä osa lisättyä todellisuutta hyödyntäviä mobiilisovelluksia. Sen avulla maailmasta on tullut käyttöliittymä, jossa liikkuminen tapahtuu konkreettisesti liikkumalla. Pokémon Go:n kiitetyin ominaisuus oli se, kuinka se sai nuoret pihalle pelaamaan. Ensimmäistä kertaa pelien historiassa peli oikeasti ja tehokkaasti motivoi liikkumaan. Tämä oli mielestäni

Pokémon Go:n paras puoli ja myyntivaltti. Sen avulla huomattiin, että mobiili lisätty todellisuus voisi olla oikeasti hyvä asia. Lisäksi sillä parannettiin pelien mainetta, koska pelien huonona puolena on pidetty pitkään pelaamisen haitallisia vaikutuksia terveydelle. Toisin sanoen pelit passivoivat ihmisiä liikaa, mutta Pokémon Go oli vastakohta, koska se suorastaan pakotti liikkumaan. Muuten et voinut liikkua pelissä.

Mobiilin lisätyn todellisuuden sovellukset vaativat paljon tarkempia sijaintitietoja kuin esimerkiksi navigointisovellukset. Navigointiin riittää ulkona sijaintitieto muutaman metrin tarkkuudella. Tämä sijaintitieto tuotetaan GPS:n (Global Positioning System) avulla. GPS on Yhdysvaltain armeijan rahoittama ja kehittämä satelliittipaikannusjärjestelmä. Lisätyn todellisuuden sovelluksetkin käyttävät GPS-järjestelmää sijaintitiedon perustana, mutta tämä ei riitä. Tällä pystytään vain muutaman metrin tarkkuuteen. Lopputarkkuus tulee käyttäjän kulkua jäljittämällä. Tarkkuus voi heittää signaalin heikkouden takia tai se voi helposti vääristyä. Tämä on aiheuttanutkin ongelmia lisätyn todellisuuden mobiilisovelluksissa. Esimerkiksi museon AR-sovelluksessa muutaman metrin heitto olinpaikassa voi merkitä paljon. Sovellus voisi esimerkiksi sekoittaa kokonaan kohteet toisiinsa ja antaa täten väärää tietoa, joka vain sekoittaisi käyttäjää. Oman haasteensa antavat myös ongelmatilanteet paikannuksen kanssa. Miten sovellus käyttäytyy, kun paikannussignaalia ei ole tai se on heikko? Matkala [2017] nostaa esille isoksi käytettävyysongelmaksi sovelluksen käyttäytymisen signaalin ollessa heikko. Matkalan käytettävyystutkimuksessa nousi ongelmaksi kohteen merkitseminen käydyksi signaalin ollessa heikko. Tämä johti siihen, että oppilaat yrittivät löytää yhtä kohtaa, jossa signaali olisi tarpeeksi voimakas. Tässä huomio siirtyi sisällöstä epäolennaiseen haitaten kohderyhmän oppimista.

Lisätyn todellisuuden sovellukset antavat uuden haasteen myös Wi-Fi-verkoissa tapahtuvalle paikannukselle. Wi-Fi-paikannusta käytetään yleensä sisätiloissa ja silloin on tärkeää myös tietää käyttäjän korkeus maanpinnasta eli missä kerroksessa käyttäjä liikkuu. Tämä on aiheuttanut uusia haasteita, koska ennen ei Wi-Fi-verkoissa tarvittu tarkkaa sijaintitietoa. Ongelmaa on pyritty ratkaisemaan kolmiomittauksen avulla. Wi-Fi-, GPS- ja matkapuhelinverkossa, jossa käyttäjä liikkuu, on useita lähettimiä. Siellä pystytään mittaamaan signaalin vahvuutta laitteeseen. Tämän tiedon avulla käyttäjän tarkka olinpaikka voidaan määrittää kolmiomittauksen avulla. Korkeutta maanpinnasta pyritään päättelemään vastaantulevan signaalin kulmasta.

2.3. Käytettävyys

Käytettävyys on laadullinen määre, joka kertoo kuinka tehokasta, vaikuttavaa ja miellyttävää järjestelmää on käyttää. Nielsenin [1994] mukaan käytettävyys koostuu opittavuudesta, tehokkuudesta, muistettavuudesta, käytön virheettömyydestä ja subjektiivisesta tyytyväisyydestä.

Mobiilin lisätyn todellisuuden sovellusten suunnittelussa on jo pitkään sovellettu yleisiä mobiilisovellusten suunnitteluperiaatteita ja käytäntöjä. Omaa suunnitteluperiaatekokoelmaa ei ole ollut [Endsley *et al.* 2017]. Sitä kuitenkin tarvitaan, sillä yleiset mobiilisovellusten suunnitteluperiaatteet eivät ole riittävän kattavia ottamaan huomioon mobiilin lisätyn todellisuuden sovellusten erityispiirteet.

Endsley ja muut [2017] tekivät tutkimuksen heuristiikoista, joita käytetään suunniteltaessa lisätyn todellisuuden sovelluksia. Tutkimuksen alussa Endsleyn ryhmä käytti lähteinään kymmentä eri heuristiikkakokoelmaa ja he kokosivat niistä AR-heuristiikkakokoelman ensimmäisen version, joka sisälsi 18 heuristiikkaa. Tutkimuksensa aikana he pyysivät eri suunnittelijoita tekemään käytettävyysarviointeja heuristiikkojen avulla. Tulokset analysoitiin samankaltaisuusdiagrammien avulla. Neljän iteraatiokierroksen jälkeen he saivat määrän kutistumaan puoleen eli 9 heuristiikkaan, jotka esittelen tässä.

Endsleyn ja muiden [2017] esittämät heuristiikat:

1. **Sovita AR käyttäjän ympäristöön ja tehtäviin:** AR-elementtien tulisi perustua reaalimaailman ja tehtäväympäristön metaforille, joista käyttäjällä on jo mielikuva.
2. **Viesti muodon avulla toiminnosta:** Virtuaalielementin muodon tulisi olla käyttäjälle tuttu, jotta käyttäjä tunnistaisi sen mahdollisuudet.
3. **Minimoi häiriöt ja ylikuormitus:** Lisätty todellisuus voi ylikuormittaa käyttäjää. Pyri minimalistiseen ja yksinkertaiseen suunnitteluun.
4. **Mukaudu käyttäjien sijaintiin ja liikkeeseen:** Järjestelmän tulisi mukautua käyttäjän liikkumiseen, kuten erilaisiin katsomiskulmiin, etäisyyksiin ja liikkeeseen.
5. **Kohdista virtuaalisen maailman elementit reaalimaailman kanssa:** Virtuaalielementtien tulisi sijaita mielekkäästi myös fyysisessä maailmassa. Tämän yhteneväisyyden tulisi jatkua ajan ja katselunäkökulman muuttuessa.
6. **Sovita käyttäjän fyysisiin kykyihin:** Vuorovaikutuksessa käytettyjen eleiden ja liikkeiden tulisi olla helppoja ja turvallisia.
7. **Sovita käyttäjän havaintokykyyn:** Lisätty todellisuus ei saa esittää tietoa, joka menee käyttäjän havaintokyvyn ylitse. Suunnittelijoiden tulisi

harkita kokoa, väriä, liikettä ja etäisyyttä suunniteltaessa lisättyä todellisuutta.

8. **Mahdollista ruudulla näkymättömien kohteiden saavutettavuus:** Suorakäyttöliittymien tulisi olla niin helppoja, että käyttäjä löytää, muistaa ja ymmärtää etsiä kohteita näkymän ulkopuolelta.
9. **Ota huomioon laitteiston kyvyt:** Lisätyn todellisuuden kokemukset tulisi suunnitella valitun laitteiston kykyjen ja rajoitusten mukaan.

Suunniteltaessa lisätyn todellisuuden mobiilisovelluksia edellä olevat heuristiikat ja suunnitteluohjeet tulisi ottaa aina huomioon. Tämän lisäksi suunnittelijan täytyy ottaa huomioon lisätyn todellisuuden sovellukselle tyypilliset suunnitteluongelmat ja erityistilanteet. Suurin osa lisätyn todellisuuden sovelluksista on suunniteltu puhelimelle, jossa näyttö ja ohjain, tässä tapauksessa ihmiskäsi, aiheuttavat oman haasteensa. Puhelimen näyttö on kohtuullisen pieni, yleensä n. 4-6 tuumaa, verrattuna tietokoneen näyttöön. Täytyy miettiä mitä voidaan näyttää ja kuinka paljon esimerkiksi tekstiä voidaan näyttää kerralla ruudulla.

Endsleyn ja muiden [2017] tekemät heuristiikat ovat kaikista laajin ja kattavin löytämistäni mobiilin lisätyn todellisuuden sovelluksia käsittelevistä suunnittelukokoelmista. Se on myös uusin niistä.

3. Tulokset

Tässä luvussa esitellään kirjallisuuskatsauksen tulokset. Ensin esitellään aineistoa ja tarkastellaan aineistossa tutkittujen sovellusten kontekseja ja sovellustyyppejä. Tämän jälkeen tarkastellaan aineistossa käytettyjä tutkimusmenetelmiä. Lopuksi aineistosta tunnistetaan käytettävyysongelmia Endsleyn ja muiden [2017] heuristiikkojen avulla.

3.1. Kirjallisuuskatsauksen aineistossa tutkitut sovellukset

Tässä kirjallisuuskatsauksessa saatiin tulokseksi kuusi kriteerit täyttävää julkaisua. Aineiston valintakriteereinä käytettiin seuraavia avainsanoja: mobiili, lisätty todellisuus, turismi, oppiminen. Aineistoon valittiin pienestä otannasta satunnaisesti kuusi artikkelia siten, että kolmen sovelluksen konteksti oli oppiminen ja kolmen sovelluksen turismi.

Aineiston julkaisut on esitelty taulukossa 1. Julkaisut on järjestetty niiden julkaisuvuoden mukaan nousevaan järjestykseen. Julkaisun yhteydessä on kuvattu julkaisussa tutkitun sovelluksen nimi ja sen aihealue. Kolmas sarake osoittaa, jos sovellus oli vielä prototyyppiasteella.

Julkaisu	Tutkittu sovellus ja sen aihealue	Prototyyppi
[Klopfer and Sheldon 2010]	Timelab 2100: oppiminen	
[Nilsson *et al.* 2012]	Astrid`s Spår: turismi	X
[Matkala 2013]	Citynomad: oppiminen	
[Müller *et al.* 2013]	GuideMe: digitaalinen käyttöohje	X
[Chen 2014]	OsloView: turismi	
[Kourouthanassis *et al.* 2015]	CorfuAR: turismi	

Taulukko 1. Tutkielman kirjallisuuskatsauksen aineisto ja aineistossa tutkitut sovellukset lyhyesti.

Aineiston sovelluksista neljä oli valmiita sovelluksia, jotka oli jo julkaistu sovelluskaupassa, kun taas kaksi sovellusta olivat vielä prototyyppiasteella. Kirjallisuuskatsaukseen valikoin tarkoituksella kolme turismisovellusta ja kolme oppimissovellusta. Vaikka aihepiirit vaikuttivat erillisiltä, kirjallisuuskatsauksen turismisovellukset olivat myös hyvin opettavaisia. Ne opettivat käyttäjilleen esimerkiksi alueen historiaa kattavasti. Olen silti rajannut ne omaksi kategoriakseen, koska niiden kohdekäyttäjät ovat erilaisia kuin oppimissovellusten. Pääerona on se, että turismisovelluksia käytetään pääasiassa viihdekäyttöön ja niiden kohderyhmä on turistit, toisin kuin oppimissovelluksissa, joissa pääasiallinen käyttö on hyötykäyttö ja pääkäyttäjäryhmänä ovat oppilaat.

Kirjallisuuskatsauksen aineiston julkaisut ovat melko tuoreita ja kaikki julkaisut ajoittuvat viiden vuoden sisälle toisistaan vuosille 2010-2015.

3.2. Aineiston tutkimusmenetelmät ja ympäristö

Seuraavaksi tarkastellaan aineistossa käytettyjä tutkimusmenetelmiä. Aineiston tutkimuksia kuvataan taulukossa 2. Julkaisut on järjestetty niiden julkaisuvuoden mukaan nousevaan järjestykseen. Julkaisun yhteydessä on esitelty julkaisun tutkimuksen ympäristö, osallistujamäärä ja tutkimussuuntaus, jota tutkimus edustaa (kvantitatiivinen/kvalitatiivinen). Kvalitatiivinen tutkimus on laadullinen tutkimus, jonka menetelmiä ovat esimerkiksi tarkkailu ja haastattelu. Kvantitatiivinen tutkimus on määrällinen tutkimus, esimerkiksi kyselytutkimus.

Julkaisu	Ympäristö	Osallistujia	Kvantitatiivinen	Kvalitatiivinen
[Klopfer and Sheldon 2010]	konteksti	20		X
[Nilsson *et al.* 2012]	konteksti	20		X
[Matkala 2013]	konteksti	37	X	X
[Müller *et al.* 2013]	laboratorio	20	X	X
[Chen 2014]	konteksti	5	X	
[Kourouthanassis *et al.* 2015]	konteksti	105	X	

Taulukko 2. Kirjallisuuskatsauksen aineiston menetelmät, osallistujat ja tutkimussuuntaus.

Suurin osa tutkimuksista tapahtui sovelluksen oikeassa kontekstissa. Tutkimuksista yksi käytti laboratoriota tutkimusympäristönä. Tutkimusympäristönä laboratorio ei pyri vastaamaan sovelluksen kontekstia. Tästä johtuen laboratoriossa on vähemmän muuttujia, jotka voisivat vaikuttaa lopputulokseen.

Osallistujamäärät olivat pääasiassa pieniä, poikkeuksena Kourouthanassiksen ja muiden [2015] tutkimus, jossa oli 105 osallistujaa. Muissa tutkimuksissa osallistujamäärät vaihtelivat 5-37 osallistujan välillä. Kaikissa aineiston tutkimuksissa osallistujat vastasivat potentiaalisia käyttäjiä. Aineistosta kaksi tutkimusta olivat kvantitatiivisia, kaksi kvalitatiivista ja kaksi molempia tutkimussuuntia edustavia.

3.3. Tutkimustulokset

Tässä kohdassa luokittelen aineiston tutkimusten tuloksia. Jaan aineiston kahteen kategoriaan sovelluksen kontekstin perusteella: (1.) turismi ja (2.) oppiminen. Esittelen kummankin kategorian yhteydessä esimerkkitutkimuksen ja tarkastelen tämän tutkimuksen kohteena olevan sovelluksen käytettävyysongelmia. Valitsemani käytettävyysongelma on yleinen kaikille tämän kategorian sovelluksille ja edustaa esimerkkitapausta.

Käytän luokittelun apuna Endsleyn ja muiden [2017] tekemiä heuristiikkoja.

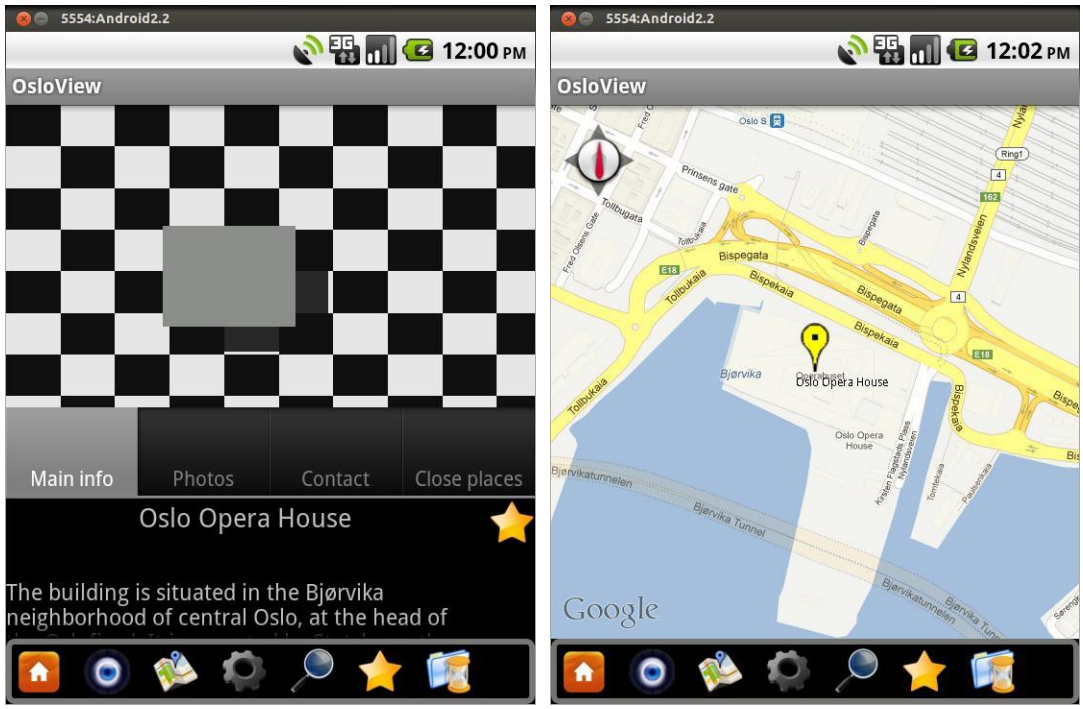
Turismisovellukset

Turismia käsittelevät Nilsson ja muut [2012], Chen [2014] ja Kourouthanassis ja muut [2015]. Ne ovat käytettävyystutkimuksia, joissa pyrittiin selvittämään tutkittavan sovelluksen käytettävyyttä ja käyttökohteita. Käytettävyystutkimusten tuloksien keruutavoissa oli eroja. Tutkimustulokset kerättiin aineiston kahdessa pelkästään kvantitatiivisessa tutkimuksessa ([Chen 2014] ja [Kourouthanassis *et al.* 2015]) kyselyiden avulla. Nilsson ja muut [2012] keräsivät tuloksensa haastatteluiden ja havainnoinnin kautta. Käytettävyystutkimukset suoritettiin sovelluksen oikeassa kontekstissa, eli oikeassa käyttöympäristössä. Näin saatiin konkreettisempi tutkimustulos, jossa otettiin huomioon myös ne muuttujat, joita tutkijat eivät osanneet edes ajatella. Esittelen nyt yhden tutkimuksen, ja tutkin tämän sovelluksen käytettävyysongelmaa, joka on yleinen lisätyn todellisuuden turismisovelluksille.

Chenin [2014] käytettävyystutkimuksessa testattiin OsloView-sovellusta (kuva 3). Sovellus oli Chenin itsensä kehittämä turistisovellus, jonka pääideana oli esitellä Oslon historiaa ja historiallisia kohteita. Tutkimukseen osallistui viisi potentiaalista käyttäjää. Käytettävyystestauksen jälkeen sovellus arvioitiin Likert-asteikkoa käyttämällä helppokäyttöisyyden ja opittavuuden osalta. Tuloksista ilmeni positiivisuutta ja kiinnostusta sovellusta kohtaan. Erityisesti ominaisuus nähdä vanhoja kuvia ja tietoja, kun sovellus tunnisti vanhan rakennuksen, oli pidetty.

Endsleyn ja muiden [2017] heuristiikkoja luokittelun apuna käyttäen käytettävyysongelmia oli erityisesti fyysisen ja virtuaalisen maailman vastaavuuksissa. Osa ikoneista ei noudattanut oikean maailman käytäntöjä, mikä aiheutti hämmennystä. Sovelluksessa ei ollut myöskään lisäkeinoja saada tietoja ikonin toiminnosta. Tämä ongelma on nähtävissä myös muissa mobiilin lisätyn todellisuuden sovelluksissa.

Ongelmia ilmeni myös käyttäjän sijainnin ja liikkeen vastaavuudessa. Chenin OsloView käyttää sijainnin määrityksenä GPS-paikannusjärjestelmää, jonka tunnettuja heikkouksia ovat epävakaus ja katvealueet. OsloView-sovellus ei ota huomioon näistä heikkouksista aiheutuvia viiveitä ja ongelmia. Erityisesti siirryttäessä lisätyn todellisuuden tilaan sovellus ei viesti käyttäjälle mitenkään käsittelevänsä GPS-sijaintitietoja. Varsinkin jos niissä on ongelmaa, käyttäjää ei informoida mitenkään pitkään aikaan. Käyttäjä ei tiedä tänä aikana, toimiiko sovellus ylipäätään.



Kuva 3. OsloView sovellus. [Chen 2014]

Oppiminen

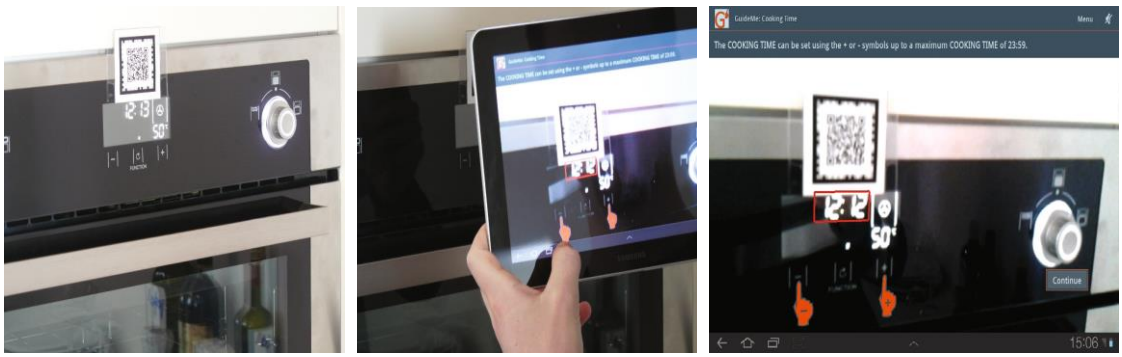
Aineiston oppimista käsittelevät tutkimukset ovat Klopfer ja Sheldon [2010], Matkala [2013] ja Müller ja muut [2013]. Ne ovat käytettävyystutkimuksia, joissa pyrittiin selvittämään tutkittavan sovelluksen käytettävyyttä ja käyttökohteita. Käytettävyystutkimusten ympäristö vaihteli. Aineistosta kaksi tutkimusta ([Klopfer and Sheldon 2010] ja [Matkala 2013]) suoritettiin sovelluksen oikeassa käyttöympäristössä, eli kontekstissa (taulukko 2), kun taas Müller ja muut [2013] keräsivät aineistonsa laboratoriossa. Tämän etuna oli se, että he pystyivät säätelemään ulkoisia muuttujia tehokkaammin ja täten mittaamaan paremmin yksittäisten muuttujien vaikutusta. Käytettävyystutkimusten tulosten keräämisessä käytettiin havainnointia ja kyselyjä. Esittelen nyt yhden tutkimuksen, jonka sovelluksen aihealueena on oppiminen ja tämän sovelluksen käytettävyysongelman, joka on yleinen tämän aihealueen sovelluksille.

Müller ja muut [2013] suorittivat käytettävyystutkimuksen GuideMe-sovellukselle. Sovellus oli Müllerin kehittämä vielä prototyyppivaiheessa ollut mobiilisovellus, joka hyödynsi lisättyä todellisuutta (kuva 4). Käytettävyystutkimuksen osana suoritettiin kaksi pienempää tutkimusta. Ensimmäisessä vertailtiin tulostetun ohjeen ja mobiilisovelluksen eroja tehtävän suorittamisen ja käytettävyyden osalta, toisessa tutkimuksessa taas video-opastuksen ja mobiilisovelluksen osalta. Tulostettu ohje osoittautui nopeimmaksi ja vähiten ongelmia aiheuttavaksi, mutta mobiilisovellus oli kuitenkin mieluisin kaikista vaih-

toehdoista, mikä johtui vuorovaikutteisuudesta ja sen tarjoamien videoiden matkimismahdollisuudesta.

Endsleyn ja muiden [2017] heuristiikkoja luokittelun apuna käyttäen käytettävyysongelmia oli erityisesti liittyen kategoriaan 5. fyysisen ja virtuaalisen maailman vastaavuus. GuideMe-sovelluksessa ei ole huomioitu kameran sijaintia tablet-laitteessa. Sovellus oletti kameran olevan keskellä laitetta, kun se oli kuitenkin laitteen päädyssä. Tämä aiheutti kuvassa siirtymän, joka hämmensi käyttäjää.

Toinen käytettävyysongelma sovelluksessa oli, että käyttöliittymä ei näyttänyt kaikkia tarvittavia tietoja, johtuen siitä, että käyttäjät pitivät tablettia liian lähellä kohdetta. Tästä johtuen kaikki tarvittava tieto ei enää mahtunut näytölle. Tämän voisi ratkaista lisäämällä näyttöön merkkejä siitä, mihin tablettia pitää liikuttaa tai merkkejä mitä tabletin näyttämän alueen ulkopuolella on.



Kuva 4. GuideMe-sovellus toiminnassa. [Müller *et al.* 2013]

3.4. Havaitut käytettävyysongelmat

Valitsin tutkimusartikkelit avainsanojen ja tutkimuksen sovelluksen aihealueen perusteella, joten kaikissa niissä ei ole nimettyjä käytettävyysongelmia. Tästä johtuen olen näissä artikkeleissa ja muissakin tutkimusartikkeleissa tulkinnut käytettävyysongelmia tutkimuksen kuvauksesta ja tuloksista. Tuloksissa esittelin tarkemmin kaksi tutkimusta ja niihin liittyviä käytettävyysongelmia. Taulukossa 3 esittelen nyt tunnistamani käytettävyysongelmat kaikista aineiston tutkimuksista. Käytettävyysongelmat ovat yleensä eritasoisia ja laajuisia, mutta tässä taulukossa en ota kantaa niiden vakavuuteen tai laajuuteen. Totean vain, että tunnistanko tämän tutkimuksen sovelluksessa tätä heuristiikkaa rikkovia käytettävyysongelmia. En testannut näitä sovelluksia itse, vaan kaikki havaintoni perustuvat tutkimusartikkeleihin.

Tutkimuskohteeni on voinut erota hyvinkin paljon tutkimuksen omasta fokuksesta, minkä vuoksi kaikkia käytettävyysongelmia ei ole kuvattu tutki-

muksesta. Näistä rajoituksista johtuen tunnistan, että tulokseni taulukossa 3 eivät ole kattavia kuvauksia sovelluksen käytettävyysongelmia.

Heuristiikat [Endsley *et al.* 2017]	[Klopfer and Sheldon 2010]	[Nilsson *et al.* 2012]	[Matkala 2013]	[Müller *et al.* 2013]	[Chen 2014]	[Kourouthanassis *et al.* 2015]
1. Sovita AR käyttäjän ympäristöön ja tehtäviin.		X				
2. Viesti muodon avulla toiminnosta.			X		X	
3. Minimoi häiriöt ja ylikuormitus.			X	X		X
4. Mukaudu käyttäjän sijaintiin ja liikkeeseen.		X	X	X	X	
5. Kohdista virtuaalisen maailman elementit reaalimaailman kanssa.	X	X			X	
6. Sovita käyttäjän fyysisiin kykyihin.						
7. Sovita käyttäjän havaintokykyyn.	X	X			X	
8. Mahdollista ruudulla näkymättömien kohteiden saavutettavuus.			X	X		X
9. Ota huomioon laitteiston kyvyt.		X	X	X		X

Taulukko 3. Havaitut käytettävyysongelmat kirjallisuuskatsauksen aineistossa.

Taulukosta 3 voidaan nähdä kaksi yleistä ongelmaa, jotka esiintyvät suurimmalla osalla sovelluksista. Peräti neljällä sovelluksella kuudesta on käytettävyysongelmia kategoriassa 4. Mukaudu käyttäjän sijaintiin ja liikkeeseen. Tämä selittyy sillä, kuinka kaikki nämä sovellukset luottavat GPS-sijaintitietoihin sijainnin määrittelyssä. Kuten edellä kerroin, kaikki on hyvin, kun GPS-sijaintitiedot toimivat, mutta entä sitten kun niissä on ongelmia? Nämä sovel-

lukset eivät olleet varautunet GPS-järjestelmän virheisiin tai viiveisiin riittävästi, ja tämä ilmeni käytettävyysongelmina.

Teknisiä ongelmia esiintyi myös paljon. Teknisiä ongelmia oli neljällä sovelluksella kuudesta. Tekniset ongelmat muuttuvat käytettävyysongelmiksi, kun ei huomioida laitteiston kykyjä riittävällä tasolla. Tämä tulee käytännössä ilmi sovelluksen epävakautena ja hitautena. Tämä tulos voi osittain johtua myös siitä, että kaksi tutkituista sovelluksista oli vielä prototyyppi asteella. Tätä ongelmaa voi ratkaista sovelluksen optimoimisella.

En löytänyt yhtään käytettävyysongelmaa kategoriaan 6. sovita käyttäjän fyysisiin kykyihin. Tämä voi johtua siitä, että suunnittelijat olivat tunteneet mobiilisovellusten suunnittelukäytäntöjä, joissa myös pitää ottaa huomioon käyttäjän fyysiset kyvyt, tai sitä ei ole kuvailtu käytettävyysongelmaksi tutkimusten kuvauksissa tai tuloksissa.

Esimerkkinä Endsleyn ja muiden [2017] heuristiikkojen kategorian 4. (Mukaudu käyttäjän liikkeeseen ja sijaintiin) rikkominen tuli hyvin ilmi Matkalan [2013] pro gradu -tutkielmassa. Matkala kuvaa kuinka oppilaat hämmentyivät sijaintia kuvaavan ikonin katoamisesta tai sen jumiutumisesta. Tämä katoaminen tai jumiutuminen johtui huonosta GPS-signaalista. Matkalan tutkima sovellus ei myöskään mitenkään viestinyt käyttäjälle tästä ongelmasta. Tämä rikkoo myös Nielsenin [1994] heuristiikkaa järjestelmän tilan näkyvyydestä.

3.5. Käyttäjäkokemus

Käyttäjäkokemus on käytettävyyttä laajempi käsite. Käyttäjäkokemus kuvaa käyttäjän reaktioita ja havaintoja hänen vuorovaikuttaessaan sovelluksen kanssa tai odotuksia sitä kohtaan. Se korostaa käyttäjän kokemusten subjektiivisuutta, kontekstisidonnaisuutta ja ajallisuutta [Olsson *et al.* 2013].

Aikaisemmat kirjallisuuskatsaukset kuten Väänänen-Vainio-Mattila ja muut [2015] ja Räty [2017] havaitsivat ongelmaksi termien käyttäjäkokemus ja käytettävyys epäselvyyden. Käytettävyys- ja käyttäjäkokemustermit ovat hyvin lähellä toisiaan ja niiden on nähty jopa limittyneen ja sekoittuneen keskenään [Väänänen-Vainio-Mattila *et al.* 2015]. Mielestäni voidaan sanoa, että käyttäjäkokemus on käytettävyyden realisoitunut tila. Tällä tarkoitan, sitä että käytettävyys vaikuttaa oleellisesti käyttäjäkokemukseen.

Räty [2017] ja Väänänen-Vainio-Mattila ja muut [2015] keskittyivät omissa tutkimuksissaan käsittelemään käyttäjäkokemusta ja sen vaikutusta suunnitteluun. Räty [2017] halusi jopa ottaa käyttäjät mukaan suunnitteluun alusta asti. Omassa tutkielmassani olen keskittynyt tähän asti käytettävyyteen tehden eron käyttäjäkokemuksesta.

Endsley ja muut [2017] kartoittivat työssään käytettävyys heuristiikkoja. Näiden heuristiikkojen rikkomisen seurauksena syntyvät käytettävyysongel-

mat eivät aina vaikuta käyttäjäkokemukseen. Ne voivat olla sellaisissa ominaisuuksissa, joita käyttäjä ei tarvitse, tai ne voivat olla esiintymättä käyttäjän käyttöympäristössä. Esimerkiksi huonosta GPS-signaalista johtuvat käytettävyysongelmat jäävät kokonaan pois alueella, jossa on hyvä GPS-signaali. Endsleyn ja muiden [2017] mukaan kuitenkin käytettävyysongelmat, jotka aiheutuvat puutteista kategoriassa 5. Kohdista virtuaalisen maailman elementit reaalimaailman kanssa, aiheuttavat aina negatiivisen käyttäjäkokemuksen.

4. Yhteenveto

Mobiilin lisätyn todellisuuden sovellukset lisääntyvät ja mielenkiinto niitä kohtaan on selkeästi kasvussa. Monet yrittävät tehdä seuraavaa Pokémon Go:n kaltaista hittiä, mutta tällä hetkellä mikään muu mobiilin lisätyn todellisuuden sovellus ei ole yltänyt samanlaisiin latausmääriin. Tämä suosion puuttuminen voi osaltaan johtua suunnittelutaidon puuttumisesta tai yksinkertaisesti hittiideoiden puuttumisesta. Suosio voi olla pieni myös teknisten vaikeuksien takia.

Teknisiä vaikeuksia mobiilin lisätyn todellisuuden sovelluksille aiheuttaa sijaintitietojen ongelmat ja huono optimointi. Sijaintitiedot mobiilin lisätyn todellisuuden sovellus saa joko GPS-sijaintijärjestelmältä tai Wi-Fi-verkolta. Molemmissa on omat ongelmansa. GPS-sijaintijärjestelmä on satelliittipaikannusjärjestelmä, jossa voi ilmetä katvealueita tai yhteysongelmia. Nämä tulisi ottaa huomioon esimerkiksi virheilmoitusten avulla. Oltaessa Wi-Fi-verkossa ongelmia voi ilmetä, kun siirrytään esimerkiksi museossa kerroksesta toiseen. Enää ei riitä paikkatietona pituus- ja leveysasteet, vaan tarvitaan myös korkeustietoja. Tämä voidaan ratkaista esimerkiksi laskemalla Wi-Fi-verkkoon tulevan yhteyden saapumiskulma.

Kirjallisuuskatsauksen aineistossa oli hyvin erilaisia lähestymistapoja mobiilin lisätyn todellisuuden sovelluksille. Joku heuristiikka tai suunnitteluohje tulee varmasti vakiintumaan käytännöksi. Nielsenin [1994] heuristiikat toimivat hyvin muissa sovelluksissa, mutta ne ovat oman aikakautensa tuote, eivätkä ota huomioon mobiilin lisätyn todellisuuden erityispiirteitä. Löytämäni Endsleyn ja muiden [2017] heuristiikkakokoelma ei ollut ainoa, mutta se oli löytämistäni jäsennellyin ja hiotuin.

Havaitsin aikaisemminkin tunnistetun käytettävyys- ja käyttäjäkokemustermien epäselvyyden myös omassa kirjallisuuskatsauksessani. Mobiilin lisätyn todellisuuden tutkimuksissa on selkeästi epäjohdonmukaisuutta käyttäjäkokemuksen määrittelemisessä. Tämä johtuu jo aikaisemmin mainitsemastani heuristiikkojen johdonmukaisen käytön puutteesta. Siksi kirjallisuuskatsaukseni suurin anti on käyttää Endsleyn ja muiden [2017] heuristiikkoja johdonmukaisesti aikaisempiin tutkimuksiin ja yleistää niissä esiintyneitä käytettävyyson-

gelmia. Kirjallisuuskatsaukseni eroaa myös aikaisemmista kirjallisuuskatsauksista siltä osin, että keskityin ainoastaan sovelluksiin, joiden aihealueena on turismi tai oppiminen.

Tämä epäjohdonmukaisuus tavoitteiden määrittelyissä ja erilaiset käytännöt mahdollistavat samojen virheiden tapahtumisen useissa eri sovelluksissa. Yksinkertaistaen: virheistä ei opita. Tämä on saanut aikaan sen, että markkinoilla suurin osa sovelluksista on vielä keskinkertaisia ja käytettävyysongelmia esiintyy laajasti. Tämä on myös vaikuttanut siihen, että Pokémon Go -pelin jälkeen ei ole tullut kuluttajien keskuudessa suosittuja mobiilin lisätyn todellisuuden sovelluksia. Jotta virheistä voisi oppia ja perusongelmat saataisiin kuriin, jonkinlaisen suunnitteluohjeiston täytyisi yleistyä. Nähtäväksi jää vakiintuuko Endsleyn ja muiden [2017] tekemä heuristiikkakokoelma normiksi vai tuleeko joku muu sen tilalle.

Viiteluettelo

Ionut Blaszkiewicz, Alexander Markowetz and Matthias Böhmer. 2017. Impact of location-based games on phone usage and movement: a case study on Pokémon Go. In: *Proc. of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services,* Article 102.

Weiqin Chen 2014. Historical Oslo on a handheld device – a mobile augmented reality application. In: *Proc. of the 18th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, 979-985.

Digi-Capital 2018. Record over $3B AR/VR investment in 2017 ($1.5B+ in Q4) Digi-capital. Published 05.01.2018. Available https://www.digi-capital.com/news/2018/01/record-over-3b-ar-vr-investment-in-2017-1-5b-in-q4/. Checked 20.11.2018.

Tristan Endsley, Kelly Sprehn, Ryan Brill. Kimberly Ryan, Emily Vincent, James Draper. 2017. Augmented reality design heuristics: designing for dynamic interactions. In: *Proc. of the Human Factors and Ergonomics Society Annual Meeting 2017*, 2100-2104.

Tobias H. Höllerer and Steven K. Feiner. 2004. Telegeoinformatics: Location-Based Computing and Services. Taylor & Francis Books Ltd.

Eric Klopfer and Josh Sheldon. 2010. Augmenting your own reality: Student authoring of science-based augmented reality games. *New Directions for Youth Development* 128, 85-94.

Thomas Olsson, Else Lagerstam, Tuula Kärkkäinen, and Kaisa Väänänen-Vainio-Mattila. 2013. Expected user experience of mobile augmented reality services: a user study in the context of shopping centres. *Personal and Ubiquitous Computing* 17, 287-304.

Panos Kourouthanassis, Costas Boletsis, CleopatraBardaki, and Dimitra Chasanidou 2015. Tourists responses to mobile augmented reality travel guides: The role of emotions on adoption behavior. *Pervasive and Mobile Computing* 18, 71-87.

Enni Mansikkamäki 2016. Pelitutkija kertoo, miksi Pokémon Go on niin suosittu Aamulehti. Saatavilla elektronisesti https://www.aamulehti.fi/kotimaa/pelitutkija-kertoo-miksi-Pokémon-go-on-niin-suosittu-24007022. (Katsottu 20.11.2018)

Jani Matkala. 2013. Toiminnan teoria kontekstitietoisen mobiilisovelluksen käytettävyyden arvioinnissa. Pro gradu -tutkielma. Informaatiotieteiden yksikkö, Tampereen yliopisto.

Paul Milgram and Fumio Kishino 1994. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems* 12, 12-18.

Pokémon Go. 2017. Pokémon Go -sovellus. Saatavilla elektronisesti Play Storesta:https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=fi (Haettu 01.12.2018)

Lars Müller, Ilhan Aslan and Lucas Krüßen. 2013. GuideMe: A mobile augmented reality system to display user manuals for home appliances. In: *Proc. of the Advances in Computer Entertainment: 10th International Conference*, 152-167

Jakob Nielsen. 1994. *Usability Engineering*. Morgan Kaufmann Pub.

Susanna Nilsson, Mattias Arvola, Anders Szczepanski, and Mganus Bång. 2012. Exploring place and direction: Mobile augmented reality in the Astrid Lindgren landscape. In: *Proc. of the 24th Australian Computer-Human Interaction Conference*, 411-419.

Sanna Räty. 2017. Näkökulma Pokémon Go:sta kuluttajasovelluksiin: Kirjallisuuskatsaus mobiilin lisätyn todellisuuden sovelluksiin ja niiden käyttötutkimuksiin. Pro gradu -tutkielma. Informaatiotieteiden yksikkö, Tampereen yliopisto.

Julian Smith. 2016. European Photo Agency (EPA). Julkaistu 12.7.2016. Saatavilla http://www.iltalehti.fi/digi/2016071221893691_du.shtml (Haettu 7.12.2018)

Inda Sofian. 2016. Beyond the Hype: A UX Reality Check on Pokémon Go. Available: https://www.uxpin.com/studio/blog/beyond-hype-ux-reality-check-pokemon-go/ (Checked on 02.12.2018)

Statista. 2018. Global revenue of the consumer mobile augmented reality app market (standalone/embedded) from 2016 to 2022 (in million U.S. dollars). Statista. Available https://www.statista.com/statistics/608990/mobile-ar-applications-installed-base-worldwide-by-type/. Checked 20.10.2018.

Tilastokeskus / Suomen virallinen tilasto. 2017. Suomen virallinen tilasto (SVT): Väestön tieto- ja viestintätekniikan käyttö [verkkojulkaisu]. Internetin käyttö mobiililaitteilla. Helsinki: Tilastokeskus. (haettu 2.12.2018).

Kaisa Väänänen-Vainio-Mattila, Thomas Olsson, and Jonna Häkkilä. 2015. Towards deeper understanding of user experience with ubiquitous computing systems: Systematic literature review and design framework. In: *Proc. of the Human-Computer Interaction*, 384-401.

Tekoälyn itseoppiminen peleissä

Joona Hautalahti

Tiivistelmä.

Tutkielmani tarkoituksena on kertoa tekoälyyn liittyvästä syväoppimisesta pelien avulla ja esitellä kaksi tutkimusta, joissa pelejä on hyödynnetty rakennettaessa itseoppivaa tekoälyä. Syväoppimisessa käytetään neuroverkkoja, joten kerron myös yleisesti neuroverkoista ja kuinka niitä on hyödynnetty näissä esimerkeissä. Toinen tärkeä osa itseoppivaa tekoälyä on evoluutioprosessi kehittyneimmän kandidaatin löytämiseksi. Esittelen tarkemmin molemmissa esimerkeissä niissä sovelletun evoluutioprosessin. Vertailemalla kahta eri tapaa pyrin löytämään tämänkaltaisen syväoppimisen hyviä ja huonoja puolia tehokkuudessa ja niiden oppimistyylissä. Tarkastelen lyhyesti myös syväoppimisesta tähän asti tehtyä tutkimusta ja minkälainen tulevaisuus tällä alueella on.

Avainsanat: tekoäly, neuroverkko, syväoppiminen, evoluutioprosessi, pelit

1.Johdanto

Tekoälyä on tällä vuosikymmenellä tutkittu paljon ja siitä löytyy lukuisia erilaisia lähestymiskohteita. Gordon [2011] määrittelee tekoälyn kokoelmaksi analyyttisiä työkaluja, jotka yrittävät imitoida elämää ja ratkaista ongelmia, mitkä ovat olleet hankalia tai mahdottomia ihmisten ratkaista. Hyvin luodulla tekoälyllä on monenlaisia käyttökohteita tulevaisuudessa, kuten esimerkiksi lääketieteen kehityksessä. Tutkielmassani tarkastelen, kuinka tekoälyä on pyritty opettamaan ja kehittämään erilaisten pelien avulla. Lähestyn tätä aihetta syväoppimisen kautta.

Tekoälyn itseoppimista kutsutaan *syväoppimiseksi* (deep learning). Syväoppimisessa hyödynnetään *neuroverkkoja* (neural network), jonka Dai ja muut [2011] määrittelevät tavaksi, jolla opetetaan tekoälyä. Standardi neuroverkko on joukko toisissaan yhdistettyjä prosessoreja, joita kutsutaan neuroneiksi. Syöteneuronit aktivoituvat ympäristöä tarkkailevien sensorien avulla ja muut neuronit niiden liitännöistä aiemmin aktivoituneisiin neuroneihin. [Schmidhuber 2015.] Luvussa 2 tutustun paremmin esimerkeissä käyttämiini neuroverkkoihin ja niiden toimintatapoihin.

Pelit ovat turvallinen ja monipuolinen alusta tekoälyn tutkimiseen. Ne ovat kautta aikain antaneet haasteita pelaajilleen, strategiaa vaativista lautapeleistä aina uusimpiin konsolipeleihin. Jokainen peli vaatii pelaajalta erilaisia tietoja ja taitoja. Tekoälyn oppimisen tutkimisessa halutaan nähdä, pystyykö tekoäly oppimaan näitä samoja taitoja, mitä ihminen oppii pelejä pelatessa. Pelit soveltuvat tähän tutkimukseen myös siksi, että ne toimivat

alustana sisältäen tietyt säännöt, joiden ympärille on helppo rakentaa tekoäly noudattamaan näitä annettuja sääntöjä.

Ensimmäinen tutkittavani peli on shakki, jossa perehdyn Fogelin ja muiden [2004] shakkiohjelmaan. Sen avulla he onnistuivat luomaan itseoppivan tekoälyn asettamalla sen pelaamaan shakkia itseään vastaan useita kertoja käyden samalla evoluutioprosessin. Kohdassa 3.1 esittelen shakin ja tekoälyn välistä tutkimusta yleisemmin. Kohdassa 3.2 kerron tarkemmin, kuinka tämä itseoppiva tekoäly oikein toimii, jonka jälkeen siirryn kohdissa 3.3 ja 3.4 sen testaukseen ja sen avulla saatuihin testituloksiin.

Toiseksi peliksi otan tarkasteluun pelin Ms. Pac-man. Pac-mania on ryhdytty hyödyntämään pelien tekoälytutkimuksessa arcade- ja konsolipelien suosion myötä. Ennen näitä pelejä tutkimus keskittyi lähinnä strategisiin peleihin kuten backgammoniin, tammeen ja toiseen tutkimaani peliin shakkiin. Esittelen Dain ja muiden [2011] luoman tekoälyn Ms. Pac-man -pelin haamulle, joka neuroverkkojen avulla kehittyy ja oppii pelin simulaatiossa. Kohdassa 4.1 esittelen pelin lyhyesti sekä kerron Pac-manin ja tekoälyn aikaisemmasta tutkimuksesta. Kohdassa 4.2 kerron, kuinka neuroverkkoja käytettiin haamun opettamisessa. Haamujen evoluutioprosessista ja sen avulla saaduista tuloksista kerron kohdissa 4.3 ja 4.4.

Luvussa 5 vertailen näitä kahteen eri peliin tehtyä tutkimusta ja vertailen niiden tehokkuutta. Tätä kautta löydän näiden tutkimuksien hyviä ja huonoja puolia. Luvun 5 loppupuolella vertailen myös tutkielmassani käytettyjä kahta hieman eri tekoälyn oppimistyyliä. Lopuksi luvun 6 yhteenvedossa pohdin, miten näitä tekoälyn itseoppimisen tuloksia ja tapoja voidaan hyödyntää ja käyttää tulevaisuudessa.

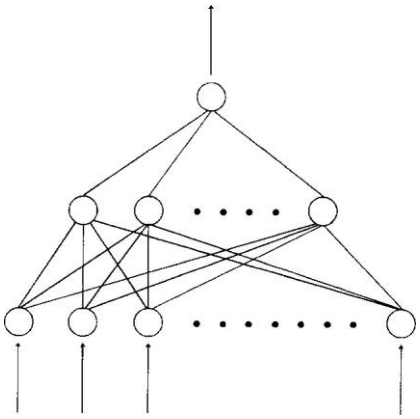
2. Neuroverkot

Eteenpäin syöttävä neuroverkko (feedforward neural network) on yksi käytetyimmistä neuroverkoista pelien tutkimuksessa. Tällainen tyypillinen neuroverkko sisältää *syöteneuroneita* (input neurons), *piiloneuroneja* (hidden neurons) ja *ulostuloneuroneja* (output neurons). Kuvan 1 alinta kerrosta sanotaan syöttökerrokseksi, ylintä kerrosta tuloskerrokseksi ja näiden kahden väliin jäävää kerrosta kutsutaan piilokerroksiksi. Näitä piilokerroksia voi olla myös useita. [Svozil *et al.* 1997]

Neuronit on yhdistetty *painoilla* (weights). Tämä paino kertoo, kuinka vahva yhteys neuroneiden välillä on ja määrittää sen, miten neuroverkko käyttäytyy ja ratkaisee ongelmat [Schmidhuber 2015]. Syöttöneuronit havaitsevat asioita paikassa, johon ne on asetettu (esimerkiksi shakkinappulat shakkilaudalla). Painoarvojen ja syöttöneuroneiden pistetulo lasketaan piiloneuronissa, jonka jälkeen se kulkee epälineaarisen muunnoksen läpi. Tämä

epälineaarinen muunnos on usein ns. *sigmoid-funktio* (sigmoid-function), mutta se voi olla myös jokin muu funktio. Käytän esimerkeissä kuitenkin sigmoid-funktiota. Sigmoid-funktio on matemaattinen funktio, joka on S-kirjaimen muotoinen käyrä ja se saa arvoja väliltä [0,1]. Sitä käytetään neuroverkkojen aktivaatiofunktiona, kun halutaan ennustaa todennäköisyys ulostulona. Todennäköisyys on myös aina olemassa välillä nollasta yhteen, joten se on hyvä funktio neuroverkkojen solmuihin. [Towards Data Science 2018] Piiloneuroneiden ulostulo prosessoidaan samalla tavalla, kunnes se saavuttaa ulostuloneuronit. Ulostuloneuronit laskevat neuroverkkojen havaittavissa olevan ulostulon. [Fogel *et al.* 2004]

Neuroverkot käyttävät myös ns. *kynnysarvoja* (biases). Kynnysarvo on arvo, jota käytetään yhdessä painoarvojen kanssa säätämään aktivaatiofunktiota. Se auttaa yleensä saavuttamaan tehokkaamman oppimisen neuroverkoilla.



Kuva 1. Eteenpäin syöttävä neuroverkko, jossa on kolme kerrosta. [Svozil *et al.* 1997]

3. Shakki ja syväoppiminen

3.1 Taustaa

Yksi keskeisimmistä haasteista tekoälyn kehityksessä on saada kehitettyä tekoälystä sellainen, joka oppii itse ilman, että ihminen liikaa vaikuttaa sen kehitykseen. Fogelin ja muiden [2004] mukaan tekoäly, joka oppii omista saavutuksista, on pysynyt saavuttamattomana. Syynä tähän on heidän mukaansa se, että on keskitytty liikaa älyllisen käyttäytymisen simuloimiseen kuin niihin mekanismiin, jotka ovat tämän käyttäytymisen takana. Fogelin ja muiden [2004] tekemät testit viittaavat siihen, että koneen oppimisalgoritmi voi kouluttaa shakkiohjelman joka pelaa suurmestarin (grandmaster) tasolla ilman, että se turvautuu tiettyihin *palkinnonanto-ongelmiin* (credit assignment problem). Tämä ongelma syntyy siitä, kun ongelmanratkaisutoiminnoissa palaute annetaan kaiken toiminnan lopussa. Tällöin oppijan täytyy osata yhdistää palaute aikaisempiin toimintoihinsa osatakseen valita oikeat toiminnot. Palkinnonanto-ongelma näkyy erityisesti *vahvistusoppimisessa*

(reinforcement learning), jossa oikeista toiminnoista palkitaan ja vääristä rangaistaan. Tämä on Fogelin ja muiden [2004] mukaan ”ontto” tapa, sillä lopputulema shakissa on heidän mukaansa epälineaarinen funktio liikkeitä molemmilta pelaajilta. Siksi yksi toiminto eli tässä tapauksessa yksi vuoro shakissa ei aina välttämättä ole oikein tai väärin. Fogel ja muut [2004] käyttävätkin testeissään systeemiä, missä pistemäärä lasketaan monen pelin perusteella, kun vertaillaan monen eri ohjelman suorituskykyä.

Shakkia käytetään paljon tekoälyn tutkimisen alustana pelin kompleksisuuden vuoksi. Yksi tekoälyä hyödyntävien shakkiohjelmien läpimurroista koettiin vuonna 1997, kun tietokoneohjelma *Deep Blue* voitti hallitsevan maailmanmestarin Garry Kasparovin. [Cambell *et al.* 2002] Tyypillisesti tällaiset shakkiohjelmat turvautuvat Fogelin ja muiden [2004] mukaan aloitussiirtojen ja loppupelin sijaintien tietokantaan, jonka avulla käytetään matemaattisia funktioita arvioimaan välisiirrot. Tämä funktio muodostuu eri ominaisuuksista kuten yksittäisen nappulan tiedoista, liikkuvuudesta, pelin temposta, kuninkaan turvallisuudesta sekä taulukosta, joka asettaa arvoja nappuloihin sen perusteella, missä ne ovat shakkilaudalla. Parametrit, jotka ohjaavat näitä ominaisuuksia, on usein asetettu ihmisen toimesta. [Fogel *et al.* 2004]

Fogel ja muut [2004] hyödyntävät ohjelmassaan kolmea neuroverkkoa, jotka arvioivat vaihtoehtoisten potentiaalisten paikkojen arvoa eri osissa shakkilautaa (yksi edessä, toinen takana ja kolmas keskellä). Tämä menettelytapa aloitettiin joukolla tekoälypelaajia, jotka alustettiin käyttämään vain saatavilla olevaa materiaalia ja paikka-arvoja avoimen lähdekoodin shakkiohjelmista. Lisäksi heitä täydennettiin antamalla nämä mainitsemani kolme neuroverkkoa, jonka jälkeen nämä tekoälypelaajat laitettiin pelaamaan toisiaan vastaan. Pelattujen pelien perusteella näitä simuloituja pelaajia arvioitiin ja osalle annettiin mahdollisuus luoda itsestään ns. jälkeläinen. Tämä eloonjäämismahdollisuus määriteltiin pelilaadun mukaan osapuolten ollessa samasta sukupolvesta. Monen sukupolven jälkeen pelaajat poimivat informaatiota pelistä ja kehittyivät sen avulla. Lopuksi parhaiten kehittynyt pelaaja laitettiin ottelemaan Chessmaster 800- ohjelmaa vastaan ja huomattiin, että kehittyneen pelaajan rating oli paljon korkeampi kuin ei-kehittyneen pelaajan. [Fogel *et al.* 2004]

3.2 Ohjelman toimintatapa

Fogelin ja muiden [2004] shakkiohjelma toimi seuraavasti: Jokainen shakkilaudan sijainti esitettiin vektorilla, jonka pituus oli 64. Jokainen komponentti tässä vektorissa kuvaa käytettävissä olevaa paikkaa laudalla. Komponentit tässä vektorissa ottivat arvonsa {-K,-Q,-R,-B,-N,-P,0,+P,+N,+B,+R,+Q,+K}, jossa 0 on tyhjä neliö ja K,Q,R,B,P ja N ovat materiaaliarvot sotilaalle (pawn), lähetille (knight), ratsulle (bishop), tornille (rook), kuninkaalle (king) ja kuningattarelle (queen). Etumerkki kertoo, onko pelinappula oma (**+**-merkki), vai vastustajan (**-**-merkki).

Pelaajan liike määriteltiin arvioimalla tulevien mahdollisten sijaintien laatua. Tämä funktio koostui kolmesta osasta:

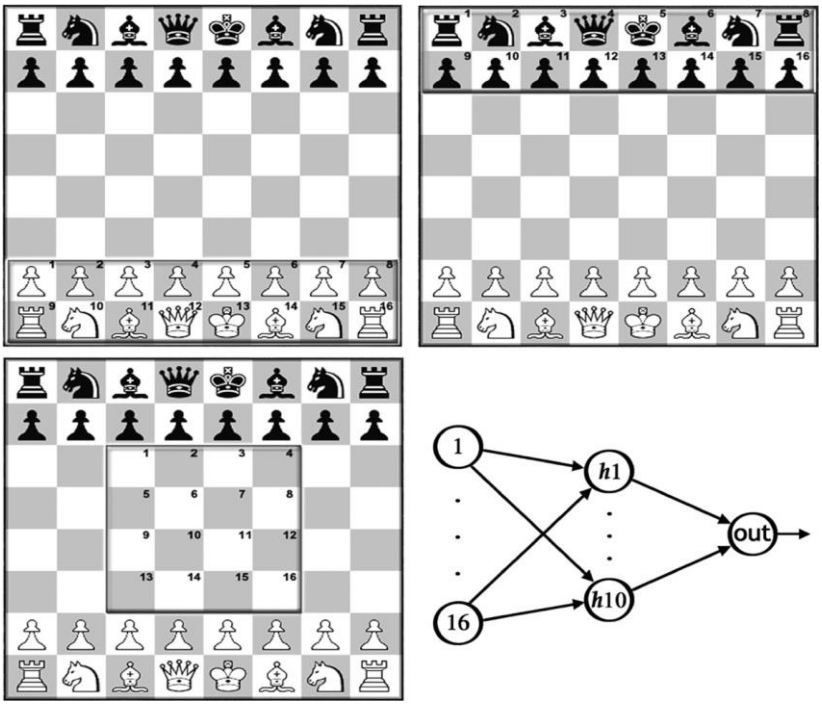
1) Materiaaliarvojen summa molemmille pelaajille
2) PVT (positional value table) eli taulukosta johdetut arvot, jotka kertovat tiettyjen nappuloiden arvot tietyissä paikoissa shakkilautaa
3) Kolme neuroverkkoa, joista jokainen sijoitettiin eri osaan shakkilautaa

Jokaisella shakkinappulalla (paitsi kuninkaalla), on PVT-arvo joka asettaa oikean arvon jokaiselle 64:lle laudan neliölle. Tämä kertoo oletetun arvon sille, että jokin nappula sijaitsee tietyssä paikassa shakkilautaa. Kuninkaalla on 3 eri PVT-arvoa, yksi sille, kun kuningas ei ole linnoitettu ja kaksi linnoittuneille kuninkaille. PVT-arvo voi olla positiivinen tai negatiivinen. Tämän on tarkoitus rohkaista tai rajoittaa pelaajaa siirtämästä pelinappulaa tiettyihin paikkoihin. [Fogel *et al.* 2004]

Pelaamiseen käytettiin *alpha-beta -hakua* (alpha-beta search) puun avulla jokaiseen laudan sijaintiin, joka katsoi tietyn verran siirtoja tulevaisuuteen. [Fogel *et al.* 2004] Alpha-beta -haku on hakualgoritmi, joka pyrkii vähentämään solmuja puusta, jos saatavilla on jo parempi vaihtoehto. Haun syvyys asetettiin neljään kerrokseen eli siirtoon, ettei ohjelman suoritusaika venyisi liian pitkäksi. Syvyyttä saatettiin pidentää, esimerkiksi shakkimattitilanteessa tai jos ollaan lähellä vastustajan pelinappulan syömistä. Paras siirto valittiin iteroimalla jokaisen kerroksen puun lehtiä riippuen siitä vastasiko kerros itse pelaajaa vai vastustajaa. Peli päättyi, kun toinen pelaajista sai shakkimatin, sama tilanne toistuu kolme kertaa pelin aikana tai pelattiin tasapeli. Tasapeli oli mahdollinen esimerkiksi tapauksissa, joissa pelaajilla oli siirtoja yli 50 tai molemmilla oli jäljellä vain kuningas. Pelin voittaja sai yhden pisteen, kun taas häviäjä menetti pisteen. Yleensä tasapelistä shakissa saa ½-pistettä, mutta tässä tapauksessa pisteitä ei tullut tai lähtenyt. [Fogel *et al.* 2004]

3.3 Testaus

Fogelin ja muiden [2004] tutkimus aloitettiin 20:llä tietokonepelaajalla. Näistä kymmenen oli vanhempia ja loput kymmenen seuraavan sukupolven jälkeläisiä. Jokaiselle annettiin nämä aiemmin mainitsemani materiaali- ja PVT-arvot sekä satunnaiset neuroverkot. Esimerkiksi kuningattaren materiaaliarvo oli 9 ja PVT välillä -40 – 80. [Fogel *et al.* 2004]



Kuva 2. Neuroverkkojen sijainti shakkilaudalla ja sen toimintapa. [Fogel *et al.* 2004]

Neuroverkot olivat täysin kytketyt syöttöverkot 16:lla tulosolmulla, kymmenellä piilosolmulla sekä yhdellä tulossolmulla. Ensimmäinen neuroverkko keskittyi kahteen etummaiseen riviin, koska sen tehtävä oli oman alueen suojaamisen tarkkailu. Toinen kolmesta neuroverkosta sijoitettiin kahteen takariviin, joka taas keskittyi vastustajan alueelle liikkumiseen. Kolmas neuroverkko laitettiin shakkilaudan keskelle tarkkailemaan keskiosan hallintaa. Kuva 2 selventää neuroverkkojen sijaintia shakkilaudalla. Fogel ja muut [2004] valitsivat kymmenen piilosolmua tähän tutkimukseen. Heidän mukaansa tulevaisuudessa tätä määrää tullaan muuttamaan neuroverkkojen kehittämiseksi. Nämä piilosolmut käyttivät standardia sigmoid-funktiota aktivaatiofunktiona. Tulossolmu käytti myös tätä samaa aktivaatiofunktiota, mutta se skaalattiin vastaamaan PVT-arvoja. Neuroverkkojen ulostulo lisättiin materiaali- ja PVT-arvoihin, jotta saatiin arvioitua jokainen mahdollinen shakkilaudan tilanne. Neuroverkkojen painot ja kynnysarvot asetettiin satunnaisesti yhtenäisen satunnaismuuttujan $U(-0.025,0.025)$ mukaisesti. [Fogel *et al.* 2004] Tässä satunnaismuuttuja U on tasajakauma, missä jokainen määrittelyjoukon arvo esiintyy yhtä todennäköisesti.

Jälkeläisen valintaan Fogel ja muut [2004] käyttivät helppoa sääntöä: kymmenen eniten pisteitä keränneet pelaajat saivat toimia vanhempina seuraavalle sukupolvelle. Jokaisesta jääneestä vanhemmasta luotiin yksi jälkeläinen mutaation avulla. Kaikki vanhempien informaatio (PVT- ja materiaaliarvot sekä neuroverkkojen paino- ja ennakkoluuloarvot) läpikävi tämän mutaation. Materiaaliarvojen mutaatioon käytettiin kaavaa $m_i^l = m_i + N(0, s_i^l)$, jossa m_i on tietyn pelinappulan materiaaliarvo vanhemmalle (i:s indeksin paik-

ka), $N(\mu, \sigma)$ on otos Gaussian satunnaismuuttujasta (keskiarvo μ ja keskihajonta σ), m_i^l on materiaaliarvo i:lle jälkeläiselle asetetulle elementille ja s_i^l on ns. strategiaparametri. Tämä strategiaparametri kehittyi käyttäen kaavaa $s_i^l = s_i \times \tau \times \exp(\mathrm{N}(0,1))$, jossa $\tau = 1/\sqrt{2n}$ ja n on kehitettävien parametrien määrä. PVT- ja materiaaliarvojen strategia-arvot asetettiin aluksi satunnaisiksi otoksiksi tasajakaumasta U, missä $U(0,0.05)$. PVT sekä neuroverkkojen arvot käyttivät mutaatiossa samaa Gaussilaista mutaatiomuotoa kuin materiaaliarvo. [Fogel *et al.* 2004]

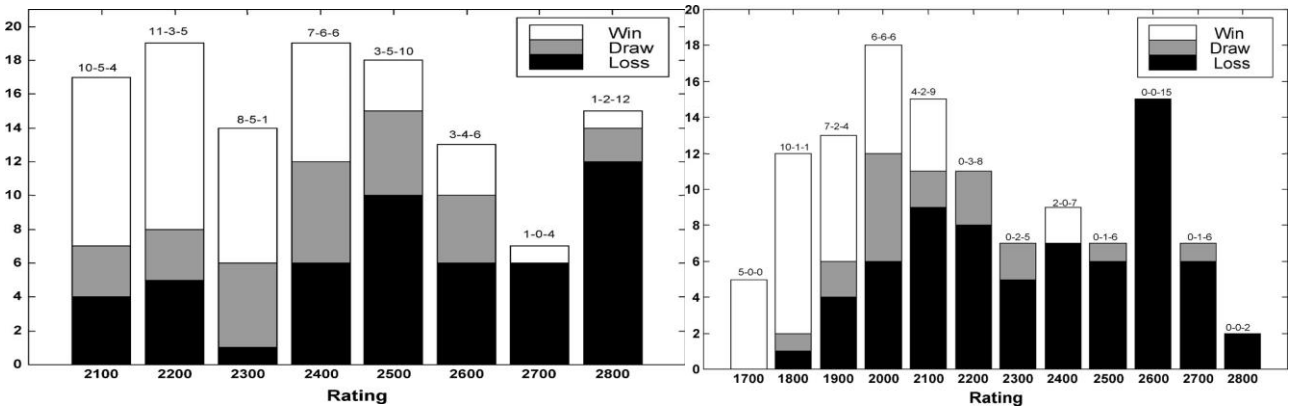
Jokainen pelaaja pelasi kymmenen peliä (viisi molempina väreinä) ja pisteet laskettiin näiden pelien yhteissummana.

3.4 Tulokset

Fogel ja muut [2004] tekivät 10 toisistaan riippumatonta tutkimusta näillä 20 tekoälypelaajalla. Jokaisessa tutkimuksessa suoritettiin 50 sukupolvea. Lopuksi jokaisen tutkimuksen parhaiten kehittynyt pelaaja laitettiin ottelemaan kehittymätöntä pelaajaa vastaan. Tutkimusten huomattiin tukevan sitä, että kehittynyt pelaaja oli parempi kuin kehittymätön pelaaja. Paras tulos tuli tutkimuksessa 8, jossa pelaajalla oli eniten voittoja kehittymätöntä vastustajaa vastaan sekä isoin ero voitoissa ja häviöissä.

Jotta tuloksia voidaan tukea vielä enemmän, Fogel ja muut [2004] pistivät tässä tutkimuksessa kahdeksan kehittyneen pelaajan ottelemaan vielä Chessmaster 8000 – nimistä shakkiohjelmaa vastaan. Yhteensä 120 peliä pelattiin ja kehittynyt pelaaja sai luokitukseksi 2437. Se vastaa luokkaa Senior Master USFC:n (United States Chess Federation) shakkiluokituksessa, ja on korkealla luokkahierarkiassa. Kehittymätön pelaaja sai Chessmasteria vastaan luokituksen 2066, joka vastaa luokkaa Expert. Se on hierarkiassa kaksi pykälää alempana kuin kehittyneen pelaajan luokka.

Kuvan 3 histogrammeista näkee, että kehittymätön pelaaja koki jo ensimmäisen tappionsa luokituksella 1800, kun taas kehittynyt neuroverkkoja käyttävä pelaaja hävisi ensimmäisen kerran vasta luokituksella 2100. Voittojen määrä korkeimmilla tasoilla oli korkeampi kehittyneellä, ja se jopa voitti korkeimmillaan ratingilla 2800 yhden pelin ohjelmaa vastaan. Fogel ja muut [2004] selittävät tätä voittoa enemmän Chessmaster-ohjelman bugina tai sen logiikkavirheenä. Heidän mukaansa tämä kehittynyt pelaaja ei kuitenkaan vastaa maailmanluokan pelaajaa.



Kuva 3. Histogrammi kehittyneen ja kehittymättömän pelaajan peleistä Chessmaster - 8000 ohjelmaa vastaan. [Fogel *et al.* 2004]

Lopuksi tämä kaikkein kehittynein pelaaja asetettiin vastakkain Pocket Friz 2.0 – nimistä shakkiohjelmaa vastaan turnausolosuhteissa. Fogelin ja muiden [2004] mukaan tämä ohjelma pelaa ratingillä 2300-2350 asetusten ollessa täydellä teholla. Yhteensä 12 peliä pelattiin, joista pelaaja voitti 9, hävisi 2 ja pelasi yhden tasoihin shakkiohjelman kanssa. Tämän perusteella voi todeta, että tämä pelaajatekoäly opetteli itse pelaamaan shakkia ja oppi strategioita, joita käytetään korkeamman tason shakkipeleissä.

4. Itseoppiva Ghost-tekoäly Pac-man pelissä

4.1 Taustaa

Kuva 4. Ensimmäinen kenttä Ms.Pac-Man -pelissä. [Dai *et al.* 2011]

Ms. Pac-man on pelinä hyvin yksinkertainen. Pelaaja ohjaa siis hahmoa nimeltä Ms. Pac-man, jonka pyrkimyksenä on syödä jokaisesta kentästä kaikki pelletit. Kentät ovat sokkelomaisia. Sokkelossa liikkuu myös haamuja, jotka yrittävät estää pelaajaa jahtaamalla tätä. Jos haamu saavuttaa Pac-manin, pelaaja häviää. Kuvassa 4 on Ms. Pac-man pelin ensim-

mäinen kenttä, josta näkee pelin sokkelorakenteen. Dai ja muut [2011] esittelivät tavan, jolla pelin haamun tekoälyä pystytään kehittämään. Perinteisesti Ms. Pac-man -pelissä haamu liikkuu perinteisen koodin (script) avulla. Dai ja muut [2011] kehittivät yhtä näistä haamuista neuroverkkojen avulla, jolloin haamu ryhtyi oppimaan itse evoluutioprosessin kautta. Tätä kuvassa 4 näkyvää pelin ensimmäistä kenttää käytettiin ainoana pelikenttänä haamun kehityksessä tutkimuksen aikana.

Pac-man on tekoälyn tutkimisen kannalta varteenotettava peli, sillä siitä löytyy monia erilaisia asioita, joita voidaan tutkia. Näitä on esimerkiksi haamujen käyttäytyminen ja yhteistyö tai optimaalisen Pac-manin luominen tekoälyn avulla. Daita ja muita [2011] ennen tutkimus onkin keskittynyt lähinnä itse Pac-maniin eikä niinkään pelin ohjaamiin haamuihin. Lucas [2005] käytti neuroverkkoja avukseen kehittäessään tekoälyä pelaamaan Ms. Pac-mania. Vuonna 2004 Yannakakis ja Hallam [2004] kehittivät neuroverkkojen avulla kontrolloidun haamun ja tutkivat sen taitoja pelata tietokoneella ohjattua Pac-mania vastaan. Heidän tutkimuksensa osoitti, että haamu pystyy sopeutumaan vastustajan strategiaan ainakin jollain asteella. Dai ja muut [2011] kritisoivat heitä siitä, että pelin versio, jossa Yannakakis ja Hallam tätä tutkivat oli paljon alkeellisempi versio Pac-manista. Kartta oli esimerkiksi paljon yksinkertaisempi ja se ei sisältänyt pelille ominaisia erikoispellettejä. [Dai *et al.* 2011] Nämä kyseiset erikoispelletit antavat siis Pac-manille mahdollisuuden hetkellisesti syödä haamuja.

Pac-manissa jokainen neljän haamun tekoälystä käyttäytyy hieman eri tavalla ja sisältää eri määrän ns. käyttäytymissääntöjä. Punainen haamu liikkuu satunnaisesti sen ollessa kaukana Pac-manista ja lähellä ollessaan se yrittää jahdata sitä. Vaaleanpunaisen haamun tekoäly käyttäytyy hieman järkevämmin, sillä se kommunikoi punaisen kanssa heidän ollessa lähellä Pac-mania. Cyan käyttäytyy lähes samalla tavalla kuin punainen, mutta kolmantena käyttäytymissääntönä se valitsee jonkun määränpään ja liikkuu suuntaan, joka on lähellä tätä sen valitsemaa määränpäätä. Oranssi on haamujen tekoälyistä tyhmin, sillä se liikkuu kentässä vain satunnaisesti. Dai ja muut [2011] valitsivat tutkittavaksi haamuksi punaisen, koska se on neljästä haamusta kaikkein aggressiivisin eikä sisällä liikaa erilaisia monimutkaisempia käyttäytymissääntöjä. [Dai *et al.* 2011]

4.2 Neuroverkkojen toimintatapa

Dai ja muut [2011] käyttivät kolmikerroksista eteenpäin syöttävää neuroverkkoa. Se päättää, miten sitä käyttävä punainen haamu liikkuu. Liikkuminen riippuu pääosin kahdesta tekijästä: missä punainen haamu ja Pac-man ovat kartalla ja voiko Pac-man syödä haamuja. Nämä kaksi tekijää Dai ja muut [2011] saivat vastaanotettua ohjelmasta ja ilmennettyä 5-ulotteisena vektorina neuroverkon sisääntulossa, joka siis sisälsi 5 muuttujaa. Tämä syöttörivi piti sisällään Pac-manin tilan sekä Pac-manin ja haamun koordinaattien ja vaa-

kasuoran korjausarvot. Neuroverkon ulostulo oli 4-ulotteinen vektori, joka sisälsi jokaisen mahdollisen suunnan, mihin haamu pystyy liikkumaan (oikea, vasen, ylös ja alas). Jokaisella suunnalla on arvo nollasta yhteen. Kun on haamun aika valita mihin se haluaa liikkua, se valitsee suunnan korkeimmalla mahdollisella arvolla. On myös mahdollista, että aiemmin kuvaamani käyttäytymissäännöt kieltävät jonkin suunnan, jolloin haamu valitsee toiseksi korkeimman arvon.

Dain ja muiden [2011] käyttämä neuroverkko sisälsi 5 neuronia piilokerroksessa. Tämä neuroverkko sisälsi siis yhteensä 15 neuronia. *Yhteyspaino* (connection weight) oli 45, painon väli oli välillä -8.192 ja 8.191 ja *verkon tarkkuus* (weight precision) 0.001. Kynnysarvo oli asetettu nollaksi ja se ei muuttunut evoluution aikana. Sigmoid-funktio toimi aktivaatiofunktiona jokaisessa neuronissa.

4.3 Evoluutioprosessi

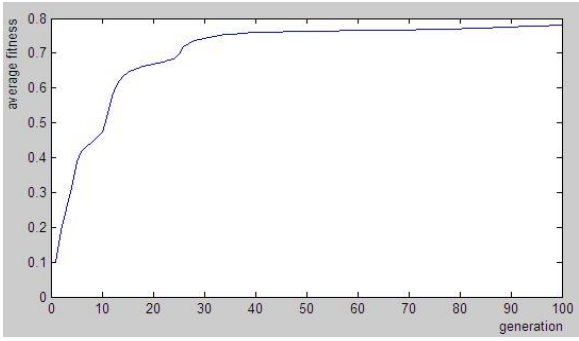
Alkupopulaatio Dain ja muiden [2011] tekoälyn evoluutioprosessissa oli 100 ja jokaiselle asetettiin satunnainen painoarvo. Tästä johtuen alkupopulaation haamut eivät osanneet jahdata Pac-mania kovin hyvin. Dai ja muut [2011] ratkaisivat tämän ongelman asettamalla samat neuroverkot jokaiselle haamulle, jolloin niiden oppimisvauhti kasvoi. Jokainen haamu lähtee hieman eri aikaan kiertämään sokkeloa, jolloin vältytään tilanteelta, jossa kaikki haamut liikkuisivat samassa kasassa.

Tilafunktion (fitness function) avulla Dai ja muut [2011] selvittivät punaisen haamun kehittymisen pelien aikana. Se on tavoitefunktio, jota käytetään yhteenvetona siitä, kuinka lähellä arvo on päämääränä asetettuja arvoja. Tämä tilafunktio keskittyy kahteen tekijään: kuinka monta kertaa haamu nappasi Pac-manin n määrä peleissä ja kuinka monta pellettiä oli kentässä jäljellä, kun Pac-man napattiin. Ensimmäinen kaava on $F = d/n$, jossa d on nappaamisien määrä ja n on pelien määrä. Toinen kaava on $L = (p - e)/p$, missä p on kaikkien pellettien määrä ja e on keskimäärä Pac-manin syödyistä pelleteistä. Yhdistämällä nämä kaksi kaavaa Dai ja muut [2011] saivat tilakaavan $C = \frac{\alpha \times F + \beta \times L}{\alpha + \beta}$, missä α ja β ovat vakioita. Parametrin α arvoksi asetettiin luku 8, sillä Pac-manin nappaaminen on merkittävämpää kuin jäljellä olevien pellettien määrä. Tämä tilakaava ei voi milloinkaan saavuttaa lukua 1, sillä haamun on mahdotonta napata Pac-man ilman, että se söisi yhtään pellettejä. [Dai *et al.* 2011]

Evoluutioprosessi tehtiin samantapaisesti kuin shakkiesimerkissä. Aluksi luotiin tietty määrä yksilöitä ja laskettiin heidän kuntoarvonsa. Pelit pelattiin simulaatiossa, jossa Pac-mania ohjasi myös tekoäly. Jokaisessa sukupolvessa jokainen yksilö pelasi 100 peliä Pac-mania vastaan. Pelien jälkeen valittiin 20 % parhaiten pärjänneistä yksilöistä, jotka risteytettiin toistensa kanssa. Risteytyksestä syntyneillä uusilla yksilöillä oli mahdollisuus suo-

rittaa mutaatio. Dai ja muut [2011] eivät kerro, mikä tuo mahdollisuus mutaatioon oli ja miten se suoritettiin. Tämän jälkeen arvioitiin uusien yksilöjen tila-arvot ja heikoimmat yksilöt korvattiin näillä uusilla (tila-arvon perusteella). Lopuksi tarkistettiin, saavutettiinko ennalta määrätty määrä sukupolvia. Jos ei, risteytettiin lisää yksilöitä, kunnes tämä määrä saavutettiin. Muuten evoluutioprosessi oli saavutettu loppuun.

4.4 Tulokset



Kuva 5. Graafi haamujen kehityksestä evoluutioprosessin aikana. [Dai *et al.* 2011]

Kuva 5 esittää, kuinka tila-arvo nousi sukupolvien noustessa. Yhteensä 100 peliä pelattiin 100:ssa sukupolvessa, jolloin yhteismäärä peleille oli 1000000. Kuvasta näkee, että kuntoarvo nousi paljon alkuarvoon verrattuna ja varsinkin alkupään oppimistahti oli varsin korkea. Alkuarvon ollessa 0,1 haamu voitti vain kymmenen peliä sadasta pelatusta pelistä. Lähestyttäessä viimeisiä sukupolvia arvo ei enää noussut niin huimaa vauhtia, mutta toisaalta haamu oli jo silloin oppinut perusstrategiat ja vain hioi niitä optimaaliseksi. Maksimiarvo on Dain ja muiden [2011] mukaan on luku 0,95. Tällöin Pac-man on ehtinyt syödä vain n. 25 % kentän pelleteistä, kun haamu nappaa sen. Viimeisessä sukupolvessa haamu sai keskimääräiseksi kuntoarvoksi 0,78. Se tarkoittaa sitä, että haamu pystyi nappaamaan Pac-manin 82:ssa pelissä 100:sta.

	Tila-arvo	**Nappaamisien määrä**	**Jäljellä olevat pelletit**
Red team	0.5065	2008	53.2
NRed team	0.5200	2027	58.8

Kuva 6. Taulukko, joka vertaa kehittyneen ja normaalin haamun tiimityöskentelyä. [Dai *et al.* 2011]

Punaisen haamun kehittyneisyyttä tarkasteltiin myös haamujen tiimityöskentelyn kautta. Dai ja muut [2011] vertasivat kahta tiimiä, jossa toisessa normaali punainen haamu oli korvattu tällä evoluutioprosessin kautta kehittyneellä punaisella haamulla. Kuvan 6 taulukko osoittaa, että kehittyneen haamun tiimi nappasi Pac-Manin useammin ja nopeam-

min kuin tiimi ilman kehittynyttä haamua. Nopeus ilmeni siinä, että Pac-Man ehti syödä vähemmän pellettejä kuin normaaleja haamuja vastaan. Pelejä tätä testiä varten pelattiin 3000.

5. Tehokkuus ja vertailu

5.1 Tehokkuuden arviointi

Fogelin ja muiden [2004] kaikista kehittynein pelaaja ei pystynyt voittamaan kaikkein tehokkaimpia shakkiohjelmia. Tekoälystä ei siis kehittynyt kuitenkaan kaikista optimaalisin tai paras pelaaja. Sama päti osittain Dain ja muiden [2011] haamuun, sillä sekään ei saavuttanut kaikista korkeinta tila-arvoa eli täysin optimaalista tapaa napata Pac-Man. Yksi syy tähän voisi olla se, että tekoälyn opettaminen peleissä neuroverkkojen avulla on melko uusi tutkimuskohde. Neuroverkkoja ja eri oppimisalgoritmeja ei siis ole saatu vielä kaikista tehokkaimmalle tasolle.

Toinen merkittävä syy on se, että tutkimusten pelien simulaatio ja tekoälyn laskentateho vie hyvin paljon aikaa. Vaikka pelit ovat toimineet alustana tekoälyn tutkimisessa jo yli 50 vuotta, vasta 2000-luvun vaihteessa ryhdyttiin huomaamaan kehitystä tällä saralla tietokoneiden tehojen noustessa. Kasparovin shakissa vuonna 1997 voittanut tekoäly *Deep Blue* laski 200 miljoonaa erilaista mahdollista siirtovaihtoehtoa sekunnissa, joka on 100 000 kertaa nopeammin kuin vuonna 1983 suurmestaritittelin voittanut shakkiohjelma *Belle.* [Fogel *et al.* 2004.] Nopeampi tietokone päätyy siis paljon parempiin lopputuloksiin kuin hitaammin laskeva tietokone. Hitaammin prosessoiva kone ei välttämättä edes löydä yhtä tehokkaita lopputuloksia. Fogelin ja muiden [2004] mukaan evoluutioprosessia olisi mahdollista nopeuttaa käyttämällä hajautettua verkkoa, jossa ohjelma hajautettaisiin monelle tietokoneelle. Nämä tietokoneet käyttäisivät pieniä ja halpoja prosessoreja.

Simulaatio Dain ja muiden [2011] tutkimuksessa kesti yhteensä 12 tuntia. Tämän aikana kone ehti pelata yhteensä miljoona peliä Ms. Pac-mania. Tutkimuksessa käytetty pelimoottori on hieman muunneltu versio normaalista Pac-manista, joka pystyy pelaamaan pelejä korkeammalla vauhdilla komentorivimuodossa. 12 tuntia oli silti melko pitkä aika, varsinkin jos se jouduttiin toistamaan useita kertoja esimerkiksi kriittisen vian tai virheen takia. Tässäkin auttaisi prosessoriltaan tehokkaampi tietokone, joka pystyisi pelaamaan nämä pelit lyhyemmässä ajassa. Tehoiltaan tehokkaampaa tietokonetta ei aina kuitenkaan ole saatavilla tai sitä ei pystytä käyttämään, jolloin täytyy vain hyväksyä pidempi prosessointiaika.

5.2 Oppimistyylin vertailu

Fogelin ja muiden shakkiohjelman oppimistyyli eroaa myös paljon Dain ja muiden haamun tyylistä. Dain ja muiden [2011] neuroverkkoja hyödyntävä tekoäly oppi paremmaksi pelaamalla tietokoneohjattua Pac-mania vastaan. Pac-man ja muut haamut eivät kehittyneet testin aikana. Pac-mania ei myöskään ohjattu manuaalisesti testin pituuden takia, vaan käytössä oli pelimoottorin koodi nimeltä SmartPac. Normaaleissa olosuhteissa tämä Pac-man pystyy voittamaan noin 33 peliä sadasta.

Fogelin ja muiden shakkiohjelma eroaa tästä siten, että siinä ohjelma pelasi omaa kehittynyttä itseään vastaan. Pelattaessa samantasoista pelaajaa vastaan oppiminen on erilaista verrattuna tilanteeseen, jossa pelaaja kehittyy koko ajan vastustajan tason pysyessä samana. Pelaajan ollessa samaa tasoa tekoälyn täytyy laskea koko ajan monimutkaisempia siirtovaihtoehtoja. Dain ja muiden haamu saattoi löytää joidenkin pelien jälkeen tavan, jolla Pac-manin sai johdonmukaisesti kiinni. Tällöin sen täytyi vain optimoida tätä tapaa, sillä vastustaja ei osannut oppia virheistään.

6. Yhteenveto

Tekoälyn itseoppiminen pelejä apuna käyttäen on iso askel sen tulevaisuutta ajatellen. Fogel ja muut [2004] pitävät shakkiohjelmansa kehitystä itsessään merkityksellisenä. Se osoittaa, että tulevaisuudessa tällaisia ohjelmia voitaisiin hyödyntää laajempaan ongelmanratkaisuun. Sillä voidaan heidän mukaansa myös löytää ratkaisuja aiemmin ratkaisemattomiin ongelmiin. Tällaiset ongelmat ovat sellaisia, joita ihminen ei ole vielä pystynyt itse ratkaisemaan itse. Itseoppinutta tekoälyä voitaisiin myös käyttää yhdessä ihmisen kanssa tehostamaan jo ihmisten saamaa tietoutta.

Fogel ja muut [2004] uskovat vahvasti siihen, että kehittynyttä tekoälyä olisi mahdollista käyttää erinäisissä asioissa tunnistamaan kahdesta eri vaihtoehdosta se parempi. Evoluutioprosessin kehittyessä tekoäly pystyy optimoimaan näitä vaihtoehtoja ja ohittamaan lopulta ihmisen näissä taidoissa. Dain ja muiden [2011] kehittynyt haamu osoittaa, että evoluutioprosessin kautta tapahtuva optimointi on mahdollista toteuttaa ainakin jo pelien tekoälyssä. Uskon kuitenkin, että tätä olisi mahdollista soveltaa myös muilla aloilla, kuten tehdaskäytössä ja lääketeollisuudessa. Nämä pelien itseoppimisessa jo käytetyt algoritmit ovat Granterin ja muiden [2017] mukaan todella käytännöllisiä esimerkiksi lääketieteellisessä tutkimuksessa, hoitosuunnittelussa sekä diagnoosien selvittämisessä. Suuret pelifirmatkin haluavat peleilleen vuosi vuodelta entistä kehittyneempiä tekoälyjä, joten pelien rintamalla kehitys jatkuu. Peleihin tulisi ainakin lisää haastetta, jos pelaajan lisäksi tekoäly osaisi kehittyä jonkin verran pelin edetessä. Näin ollen tekoälyn itseoppimista tullaan tutkimaan ja hyödyntämään varmasti vielä tulevaisuudessakin.

Viiteluettelo

Murray Campbell, A. Joseph Hoane Jr and Feng-hsiung Hsu. 2002. Deep Blue. *Artificial Intelligence* 134, 1, 57-83.

Jia-Yue Dai, Yan Li, Jun-Fen Chen and Feng Zhang. 2011. Evolutionary neural network for ghost in Ms. Pac-Man. In: *Proc. of the International Conference on Machine Learning and Cybernetics* 2, 732-736.

David B. Fogel, Timothy J. Hays, Sarah L. Hahn and James Quon. 2004. A self-learning evolutionary chess program. *IEEE Proceedings* 92, 12, 1947.

Scott R. Granter, Andrew H. Beck, David J. Papke Jr. 2017. AlphaGo, deep learning, and the future of the human microscopist. *Arhives of Pathology & Laboratory Medicine* 141, 5, 619-621

Brent M. Gordon. 2011. *Artificial intelligence: Approaches, Tools and Applications*. Nova Science Publishers.

Simon M. Lucas. 2005. Evolving a neural network location evaluator to play ms.pac-man. In*: Proceedings of the IEEE Symposium on Computational Intelligence and Games* 203-210.

Jürgen Schmidhuber. 2015. Deep learning in neural networks: an overview. *Neural Networks* 61, 85-117.

Daniel Svozil, Vladimír Kvasnicka, Jiří Pospíchal. 1997. Introduction to multi-layer feed-forward neural networks. *Chernometrics and Intelligent Laboratory Systems* 39, 1, 43-62.

Georgios N. Yannakakis, John Hallam. 2004. Evolving opponents for interesting interactive computer games. In: *Proceedings of the 8th International Conference on the Simulation of Adaptive Behaviour*, 499-508.

Towards Data Science. 2018. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Checked 17.11.2018.

Verisolujen segmentointimenetelmät

Jere Huumonen

Tiivistelmä.
Monien sairauksien diagnosointi perustuu verinäytteeseen, josta voidaan esimerkiksi laskea verisolujen määrät mikroskoopin avulla, ja sen perusteella tehdä päätelmiä mahdollisista sairauksista. Nykyään tämä prosessi toimii usein automaattisesti, joten tarvitaan kehittyneitä tietokoneistettuja tekniikoita, jotka takaavat prosessin tarkkuuden. Yksi tärkeimmistä vaiheista solukuvien analysoimisessa on kuvan segmentointi, jossa kuvaa yksinkertaistetaan siten, että kuvassa olevat solut pystytään erottelemaan taustasta. Vaihe on tärkeä, koska myöhempien vaiheiden tulos riippuu vahvasti segmentoinnin onnistumisesta. Verisolujen segmentointiin liittyy paljon ongelmia, kuten solujen päällekkäisyydet. Usein sovellukseen liittyvät ongelmat määrittävät millaiset menetelmät toimivat parhaiten segmentoinnissa. Tämän vuoksi erilaisia menetelmiä on lukuisia. Tässä tutkielmassa perehdytään joihinkin yleisimpiin segmentointimenetelmiin, joita verisolujen segmentoinnissa käytetään. Tarkastelun kohteena on erityisesti kynnystys, klusterointi ja watershed -menetelmä, mutta myös muita menetelmiä sivutaan.

Avainsanat ja -sanonnat**:** kuvansegmentointi, segmentointimenetelmät, verisolut, kynnystys, klusterointi, watershed.

1. Johdanto

Lääketieteessä solukuvien analysoimisessa käytetyt tietokoneistetut tekniikat ovat syrjäyttäneet manuaaliset keinot viimeisten vuosikymmenien aikana. Alalla eteenpäin pääseminen riippuu yhä enemmän näiden tekniikoiden hallitsemisesta ja niiden kehittämisestä.

Solujen analysoiminen on tärkeä tehtävä, ja sillä on paljon erilaisia käyttökohteita. Sitä tarvitaan erityisesti solujen tieteellisessä tutkimisessa, jossa soluja tutkimalla pyritään ymmärtämään niiden rakennetta ja toimintaa paremmin. [Bengtsson 2003]. Hyvin yleinen käyttökohde on kuitenkin erilaisten sairauksien diagnosointi, jossa solujen tutkimisella on merkittävä rooli. Soluja ja kudoksia tutkimalla voidaan diagnosoida ja ennaltaehkäistä erilaisia sairauksia, kuten syöpää, joka on yleisin tutkimuskohde solututkimuksen alueella [Bengtsson 2003]. Syövän tutkimisen lisäksi toinen merkittävä sairauksien diagnosointiin liittyvä käyttökohde on verisolujen tutkiminen, johon tässä tutkielmassa perehdytään.

Automaattista solujen analysointia on tutkittu jo 1950-luvulta lähtien [Bengtsson 2003]. Solujen analysointiin kuuluu useita erilaisia vaiheita, joihin kaikkiin liittyy omat ongelmansa. Tässä tutkielmassa perehdytään kuitenkin kuvan segmentointiin, joka on yksi tärkeimmistä vaiheista. Segmentointi rajataan verisoluihin, koska se on huomattavasti selkeämpi kuin muut solututkimuksen alueet. Tutkielmassa keskitytään pääosin segmentoinnin menetelmiin, joita verisolujen segmentoinnissa käytetään. Kaikkia menetelmiä ei kuitenkaan käsitellä, vaan ainoastaan joitain yleisimpiä.

Luvussa 2 käsitellään segmentoinnin taustaa ja sen käyttöä sekä ongelmia verisolujen kohdalla. Luvusta 3 lähtien käsitellään joitain yleisempiä segmentointimenetelmiä, niiden toimintaa ja käyttökohteita verisolujen segmentoimisessa. Luku 3 käsittelee kynnystystä, luku 4 käsittelee klusterointia, luku 5 käsittelee watershed -menetelmää ja luvussa 6 käsitellään lyhyesti muita menetelmiä. Luku 7 on yhteenveto.

2. Taustaa

2.1 Automaattinen solujen analysointi

Ennen solujen analysointi tapahtui pääosin manuaalisesti, mutta nykyään käytetään paljon automatisoituja järjestelmiä [Bengtsson 2003]. Manuaalisessa analysoinnissa näytettä tutkitaan mikroskoopin avulla, ja päätelmät tehdään perustuen ihmisen tekemiin havaintoihin. Tällaiseen manuaaliseen tutkimiseen liittyy kuitenkin paljon ongelmia. Analysointiprosessi on manuaalisesti hyvin aikaa vievä prosessi ja analysoinnin lopputulos on hyvin riippuvainen näytteen analysoivan henkilön kyvyistä, kuten tietämyksestä ja näkökyvystä [Huang *et al*. 2012]. Tarkkuus saattaa kärsiä esimerkiksi väsymyksen seurauksena, jolloin virheitä syntyy herkemmin. Manuaalisten menetelmien tuottamat tulokset ovat myös hyvin subjektiivisia ja niitä voi olla vaikea toistaa [Dorini *et al.* 2013]. Näiden syiden vuoksi alalla ollaan siirrytty prosessin automatisointiin, joka on tapahtunut laajasti parin viimeisen vuosikymmenen aikana tietokoneiden yleistyttyä ja lääketieteen ammattilaisten totuttua niiden käyttöön [Bengtsson 2003].

Automaattiset järjestelmät kuitenkin ratkaisevat monta ongelmaa. Ne erityisesti vähentävät ihmisen tekemän virheen mahdollisuutta ja takaavat paremman tarkkuuden [Huang *et al*. 2012]. Kokonaan automaattiset järjestelmät ovat kuitenkin melko vaikeita toteuttaa, koska tällöin tietokoneen täytyy pystyä suorittamaan koko prosessi lähtien näytteen asettamisesta aina havaintojen tekemiseen. Erilaisia vaiheita on lukuisia, ja osa niistä on vaikeampia tietokoneelle kuin ihmiselle. Esimerkiksi ihminen pystyy helposti tunnistamaan verisolut

ja laskemaan ne kehittyneen silmänsä ansiosta, mutta tietokoneelle sama tehtävä ei ole yhtä helppo.

Kuvanprosessoinnin tärkein vaihe on kuvan segmentoiminen. Se tarkoittaa prosessia, jossa kuvan esitystapaa muutetaan ja kuva jaetaan erilasiin alueisiin. Sillä on kaksi tavoitetta. Ensimmäinen tavoite on jakaa kuva erilaisiin alueisiin, jotka eroavat toisistaan, esimerkiksi värin perusteella. Toinen tavoite on muokata kuvan esitystapaa siten, että se muodostaa merkittävämmän kokonaisuuden myöhempää analysointia varten. [Shapiro and Stockman 2001.] Segmentoinnin tarkoitus on erityisesti yksinkertaistaa kuvaa, jotta se pystyttäisiin analysoimaan helpommin joko ihmisen tai tietokoneen toimesta (kuva 1). Yleisesti ottaen kuvan segmentointia käytetään paljon lääketieteessä, mutta myös monilla muilla alueilla, kuten konenäössä, kasvojen tunnistuksessa, liikenteenohjausjärjestelmissä ja satelliittikuvien analysoimisessa.

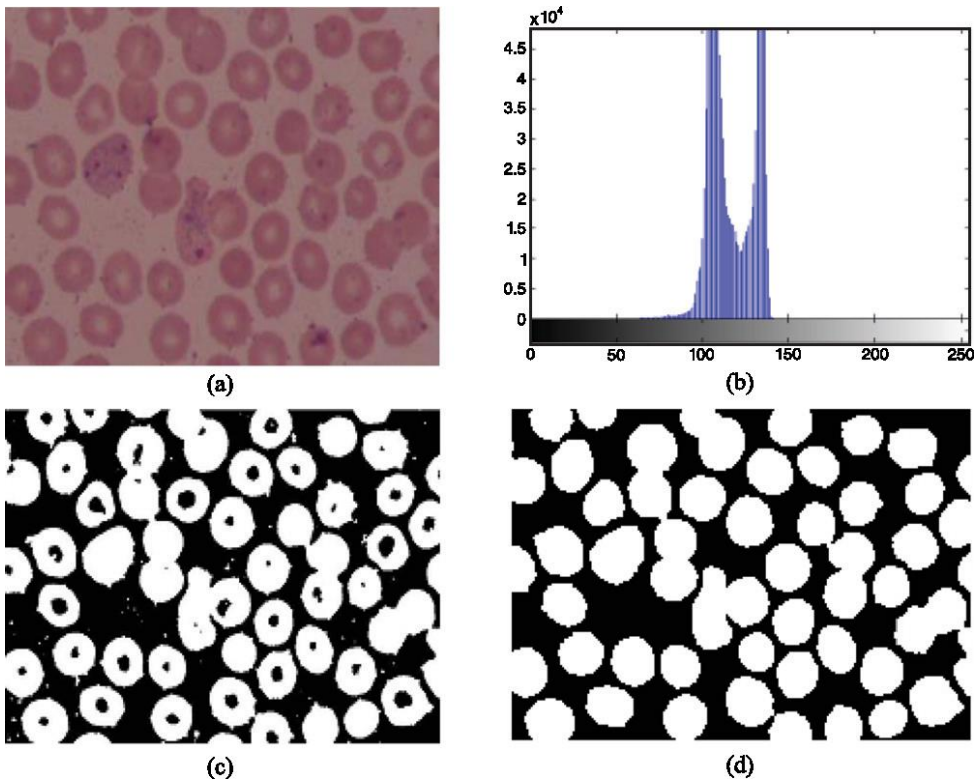
2.2 Verisolujen segmentointi

Ennen kuin verisolujen segmentointimenetelmiä voidaan tarkastella, ensin täytyy ymmärtää verisolujen toiminta ja niiden segmentointiin liittyvät ongelmat. Suurin osa verestä koostuu nestemäisestä veriplasmasta. Veriplasman lisäksi veri sisältää verisoluja, jotka jakautuvat pääosin punasoluihin eli erytrosyytteihin, valkosoluihin eli leukosyytteihin ja verihiutaleisiin eli trombosyytteihin. Nämä solut eroavat toisistaan ominaisuuksiltaan, kuten koon ja värin perusteella. Jokaisella verisolulla on oma tehtävänsä. Valkosolut toimivat osana kehon immuunijärjestelmää. Niiden tehtävä on puolustaa kehoa infektioita vastaan. Epänormaalin korkea valkosolujen määrä saattaa kertoa esimerkiksi infektiosta tai leukemiasta. Punasolujen päätehtävä on puolestaan kuljettaa happea. Liian vähäinen punasolujen määrä saattaa viitata anemiaan. Liian korkea määrä saattaa puolestaan lisätä sydänkohtauksen tai aivoverenkiertohäiriön riskiä. [Saronde 2018.] Laskemalla verinäytteestä sen sisältämien verisolujen määrät, pystytään siis määrittämään sairauksia. Tämän vuoksi verikokeiden käyttö on hyvin yleistä.

Verisolujen analysoimisessa niiden segmentoinnin päätavoite on erotella verisolut taustasta (kuva 1). Segmentoiduista verisoluista tehdään sitten myöhemmissä vaiheissa havaintoja, esimerkiksi niiden kokonaismäärän laskemisen perusteella. Punasolujen ja valkosolujen segmentointi eroaa hiukan toisistaan. Yleisesti punasolujen segmentointi on paljon helpompaa kuin valkosolujen segmentointi, sillä punasoluilla ei ole ollenkaan tumaa. Toisaalta punasoluja on myös paljon enemmän ja ne ovat myös pienempiä kuin valkosolut, mikä saattaa vaikeuttaa niiden segmentointia. Valkosoluja sen sijaan on useita eri tyyppejä. Ne jakautuvat viiteen eri ryhmään ja eroavat niiden tekstuurin, värin,

koon ja tuman sekä sytoplasman morfologian perusteella [Hamghalam *et al.* 2009]. Usein valkosolut halutaan erotella niiden tyypin perusteella, mikä vaatii hyvin tarkkaa segmentointia, koska tumat ovat melko pieniä. Valkosolujen ongelma on myös niiden vaalea väri. Ennen segmentointia ne pitääkin ensin värjätä, koska muuten niitä on hyvin vaikea havaita [Hamghalam *et al.* 2009]. Valkosoluja on myös huomattavasti vähemmän kuin punasoluja. Tavallisesti suhdeluku on noin 1 valkosolu jokaista 600-700 punasolua kohden [Saronde 2018], mutta niiden määrät solukuvissa kuitenkin vaihtelevat.

Verisolukuvien segmentointiin liittyy yleisesti paljon erilaisia käytännön ongelmia. Solujen sijainti ja kohteiden suuri määrä kuvaushetkellä voi olla ongelmallinen. Soluista saattaa muodostua ryppäitä, joista yksittäisiä soluja voi olla vaikea segmentoida. Tällaiset tilanteet johtuvat yleensä korkeasta punasolujen ja valkosolujen määrästä. Kohinaa saattaa syntyä kuvan muodostamisen ja solujen värjäytymisen yhteydessä, jolloin kuvaa pitää usein suodattaa, esimerkiksi käyttämällä mediaanisuodinta. Näiden lisäksi myös kuvien valaistuksen ja värjäytymisen epäjohdonmukaisuudet saattavat vaikeuttaa segmentointiprosessia. [Mohapatra *et al*. 2011.] Kuvien segmentointiin ei ole olemassa yleistä ratkaisua, joten erilaisia segmentointimenetelmiä on olemassa valtava määrä. Seuraavissa luvuissa perehdytään yleisimpiin solukuvien segmentoinnissa käytettyihin menetelmiin.



Kuva 1. a) Alkuperäinen kuva. b) Histogrammi pikselien harmaansävyistä. c) Segmentoitu kuva. d) Lopullinen tulos suodatuksen jälkeen. [Devi *et al.* 2017.]

3. Kynnystys

3.1 Yleistä

Verisoluista otetun kuvan tausta ja siihen kuuluvat solut erottuvat yleensä hyvin helposti toisistaan. Värjäyksen seurauksena verisolut näkyvät värillisinä ja tausta pysyy vaaleana. Tätä eroa kuvan taustan ja siinä olevien kohteiden välillä voidaan hyödyntää segmentoinnissa. *Kynnystys* (thresholding) on yksinkertaisin segmentointimenetelmä. Sen avulla voidaan erottaa kohteita kuvasta määrittelemällä kuvan kukin pikseli joko osaksi *taustaa* (background) tai osaksi *etualaa* (foreground). Etualalla tarkoitetaan kuvassa olevia kohteita, jotka halutaan erotella taustasta. [Sezgin and Sankur 2004.] Esimerkiksi solukuvissa etualalla tarkoitetaan soluja. Taustalla tarkoitetaan puolestaan kuvan merkityksetöntä osaa, josta solut erotellaan.

Erottelu taustan ja etualan välille perustuu yleensä pikselien harmaansävyisyyteen. Tämän vuoksi kuvat esitetään yleensä harmaansävyisinä. Erottelua varten tarvitaan kynnysarvo T, jota vertaillaan pikseleiden arvoihin. Kynnyksiä on joko yksi tai useampi, riippuen menetelmästä. Pikselien arvoista voidaan myös muodostaa histogrammi, jonka perusteella harmaansävyisyyden jakautuminen pystytään paremmin havaitsemaan, ja sen perusteella tehdä mahdollisia päätelmiä kynnysarvosta (kuva 1b). Kynnyksen T avulla pikselit voidaan jakaa taustaan ja etualaan. Pikselin arvon ollessa pienempi kuin T, se määritetään osaksi etualaa. Sen sijaan pikselin arvon ollessa suurempi kuin T, pikseli määritetään osaksi taustaa. [Sezgin and Sankur 2004.] Kynnystetystä kuvasta muodostuu binäärikuva, jossa mustista pikseleistä koostuu kuvan tausta ja valkoisista pikseleistä kuvassa oleva kohde. Tällöin kukin pikseli kuuluu joko etualaan tai kuvan taustaan.

Yksinkertaisuutensa vuoksi kynnystys on hyvin suosittu menetelmä moniin segmentointiongelmiin. Sille onkin kehitetty lukuisia erilaisia menetelmiä, jotka eroavat pääosin kynnysarvon valitsemisen perusteella. Sezgin ja Sankur [2004] jaottelevat kynnystysmenetelmät kuuteen eri kategoriaan:

1. Histogrammin muotoon perustuvat menetelmät, jossa kynnysarvot määritellään tarkastelemalla kuvan pikseleistä muodostuvan histogrammin muotoja, kuten huippuja ja kaaria.
2. Klusterointiin perustuvat menetelmät, jossa pikselit jaetaan kahteen ryhmään, perustuen histogrammin lohkoihin, jotka kuvaavat kuvan etualaa ja taustaa. Klusteroinnilla tarkoitetaan aineiston jaottelua samankaltaisuuden perusteella.

3. Entropiaan perustuvat menetelmät, jossa pikselit jaetaan etualaan ja taustaan perustuen harmaan sävyjen jakautumisen entropiaan. Kuvien kohdalla entropialla mitataan kuvan pikselien epäjärjestystä, mikä perustuu pikselien jakautumiseen. Kahdella eri kuvalla voi olla kuitenkin samanlaiset histogrammit, mutta erilainen entropia.
4. Piirteiden samankaltaisuuteen perustuvat menetelmät, jossa alkuperäistä kuvaa ja siitä muodostettua binäärikuvaa vertaillaan ja pyritään löytämään yhtäläisyyksiä.
5. Spatiaaliset menetelmät, jossa hyödynnetään korkeamman asteen todennäköisyysjakaumia ja/tai korrelaatiota kuvan pikseleiden välillä.
6. Paikalliset menetelmät, jossa jokaiselle pikselille määritetään oma kynnysarvo, joka saadaan määritettyä pikselin naapuruston avulla.

3.3 Otsun kynnystys

Otsun kynnystys [Otsu 1979] on yksi tehokkaimmista ja yleisimmistä automaattisista kynnystysmenetelmistä. Se on klusterointiin perustuva kynnystysmenetelmä. Sen tavoitteena on löytää kynnysarvo, joka minimoi luokkien, eli etualan ja taustan, sisäisen varianssin tai maksimoi luokkien välisen varianssin. Tämän kynnyksen avulla tehdään lopullinen jako taustan ja etualan pikseleiden välillä. Sisäinen varianssi saadaan laskettua seuraavalla kaavalla [Morse 2000; Otsu 1979]:

$$\sigma^2_{sisäinen}\,(T) = n_1(T)\,\sigma_1^2\,(T) + n_2(T)\,\sigma_2^2\,(T), \qquad (1)$$

kun,

$$n_1(T) = \sum_{i=0}^{T-1} p(i)$$

$$n_2(T) = \sum_{i=T}^{N-1} p(i)$$

ja

$\sigma_1^2\,(T)$ ja $\sigma_2^2\,(T)$ = Kynnysarvon alapuolella (tausta) ja yläpuolella (etuala) olevien pikselien varianssi.

Sisäisen varianssin laskeminen jokaiselle mahdolliselle kynnysarvolle on kuitenkin työläs prosessi. Helpompi tapa on laskea luokkien välinen varianssi vähentämällä luokkien sisäinen varianssi kokonaisvarianssista. Luokkien välinen varianssi voidaan laskea seuraavalla kaavalla [Morse 2000; Otsu 1979]:

$$\sigma^2_{välinen}(T) = N_1(T)\, N_2(T)[\, \mu_1\,(T) - \,\mu_2\,(T)]^2, \qquad (2)$$

kun μ_1 ja μ_2 ovat luokkien pikselien arvojen keskiarvot. Otsun kynnystyksen algoritmi on melko yksinkertainen. Jokaiselle kynnykselle *T* suoritetaan seuraavat vaiheet [Morse 2000]:

1. Jaetaan pikselit kynnysarvon perusteella luokkiin.
2. Lasketaan molempien luokkien keskiarvo.
3. Lasketaan luokkien keskiarvojen erotuksen neliö.
4. Kerrotaan molempien luokkien pikselit yhteen ja kerrotaan siitä saatu tulos edellisen vaiheen tuloksella.

Lopulta valitaan kynnys, jolla saatiin edellisestä vaiheesta suurin arvo. Tämän kynnyksen avulla kuva segmentoidaan taustan ja etualan välille.

Edellä esitelty Otsun kynnystys ei ole kuitenkaan sopivin menetelmä kaikkiin tilanteisiin sellaisenaan. Se on niin sanottu globaali kynnystysmenetelmä, jossa määritellään vain yksi kynnysarvo, jolla pikselit luokitellaan. Tähän liittyy kuitenkin ongelmia. Jos kuvassa olevien kohteiden pinta-ala on pieni taustaan verrattuna, segmentoinnista ei välttämättä tule kovin tarkka. [Huang *et al.* 2012.] Tällaisissa tilanteissa yksi ratkaisu on käyttää aiemmin mainittua paikallista kynnystystä, jossa kynnysarvo muuttuu riippuen pikselin naapurustosta. Tällainen tilanne saattaa olla esimerkiksi valkosolujen segmentoinnissa, jossa tarkoitus on eritellä kuvassa olevat valkosolut niiden sisältämän tuman perusteella, mikä ei kuitenkaan onnistu tavallisessa Otsun kynnystyksellä, jos tuma-alue on pieni kuvan muuhun alueeseen verrattuna [Huang *et al.* 2012].

3.4 Käyttökohteet

Kynnystystä on käytetty hyvin paljon verisolujen segmentoinnissa. Kynnystystä käytetään kuitenkin vain harvoin sellaisenaan. Yleensä sitä käytetään yhtenä vaiheena segmentoinnissa. Kuva saatetaan esimerkiksi segmentoida käyttäen kynnystystä, ja siitä saadun tuloksen perusteella käyttää jotain muuta menetelmää. Kynnystys toimii kuitenkin hyvänä ja helppona tapana tuottaa binäärikuvia. Tällaista menettelyä käytti esimerkiksi Hamghalam ja muut [2009]. He ensin segmentoivat valkosolun tuman käyttäen Otsun kynnystystä, ja tämän jälkeen erottelivat sen tarkemmin aktiivisten reunamallien (katso luku 6) avulla.

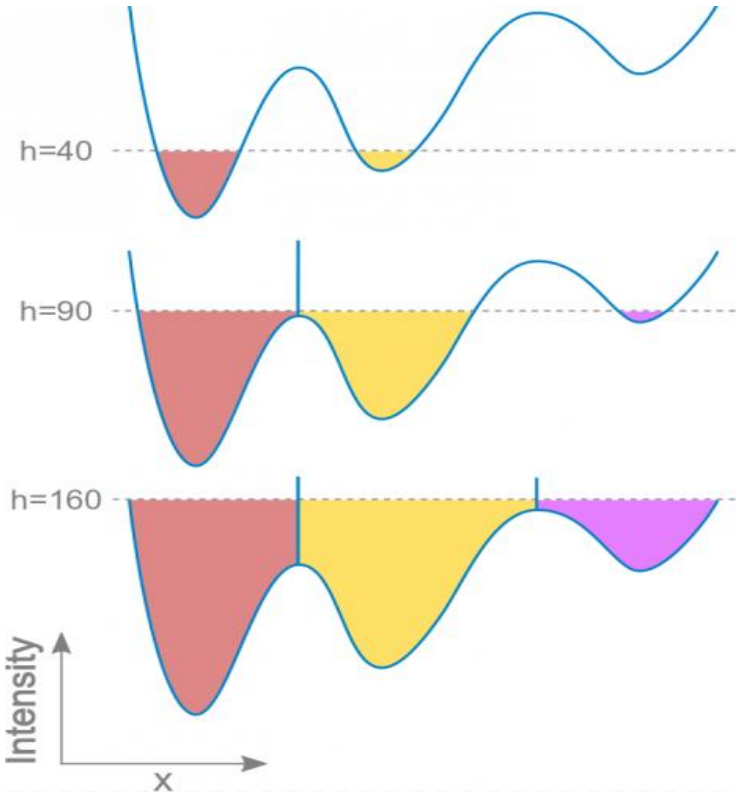
Huang ja muut [2012] käyttivät Otsun monitasoista kynnystystä (multilevel thresholding) valkosolujen tumien segmentoimiseen. Monitasoisessa kynnystyksessä voidaan valita useampi kynnys, jolla segmentointi suoritetaan. Tutkimuksessa käytettiin erikokoisia solukuvia, joten jokaiselle kuvalle pystyttiin määrittämään erilainen määrä kynnyksiä riippuen kuvan koosta, mikä takaa segmentointitarkkuuden erikokoisilla kuvilla.

4. Watershed-menetelmä

Watershed-menetelmä on matemaattiseen morfologiaan perustuva segmentointimenetelmä, joka alun perin kehiteltiin jo 1970-luvulla. Menetelmän erityispiirre on se, että se tuottaa aina rajat eri alueiden välille. Esimerkiksi solukuvissa osittain päällekkäiset ja toisissa soluissa kiinni olevat solut erottuvat toisistaan rajan avulla käytettäessä menetelmää, jolloin solut pystytään erittelemään ja laskemaan helpommin. Tämän ominaisuuden vuoksi menetelmää käytetäänkin yleisesti erottelemaan kohteita toisistaan.

Watershed (vedenjakaja)-menetelmän nimitys tulee maantieteestä. Sillä tarkoitetaan valuma-alueiden välistä maastoa, jossa maahan valunut vesi valuu eri vesistöihin perustuen maanpinnan topografiaan. Menetelmässä kuvan voidaan ajatella olevan topografinen gradienttikuva, jossa eri harmaansävyn arvot määrittävät kuvan pikselien korkeudet. Vaaleiden pikselien voidaan ajatella muodostavan korkeita harjanteita, joita kutsutaan vedenjakajiksi (watershed ridge lines), ja tummista pikseleistä voidaan ajatella muodostuvan pohja-alueita, joista pienimpiä kutsutaan paikallisiksi minimeiksi (local minima). Harjanteiden ja paikallisten minimien välissä puolestaan on jyrkänteitä, jotka koostuvat tummien ja vaaleiden pikselien välisistä arvoista. Jyrkänteet ja minimit muodostavat valuma-alueita (catchment basin). [Tulsani *et al*. 2013.]

Paikallisiin minimikohtiin, eli kohtiin, joissa pienimmät pikseliarvot sijaitsevat, alkaa muodostumaan vettä peittäen kaikki minimikohdat. Veden nousu jatkuu tasaisesti peittäen jyrkänteitä ja käyden läpi kaikki harmaansävyn arvot järjestyksessä arvosta 0 lähtien. Kun vesi saavuttaa harjanteen, eli vedenjakajan, siihen muodostetaan pato, joka jää vedenpinnan yläpuolella. Lopullinen segmentointi saadaan, kun vesi peittää kaiken muun paitsi padot, jolloin padoista muodostuu segmentointi (kuva 2). [Tulsani *et al*. 2013.]

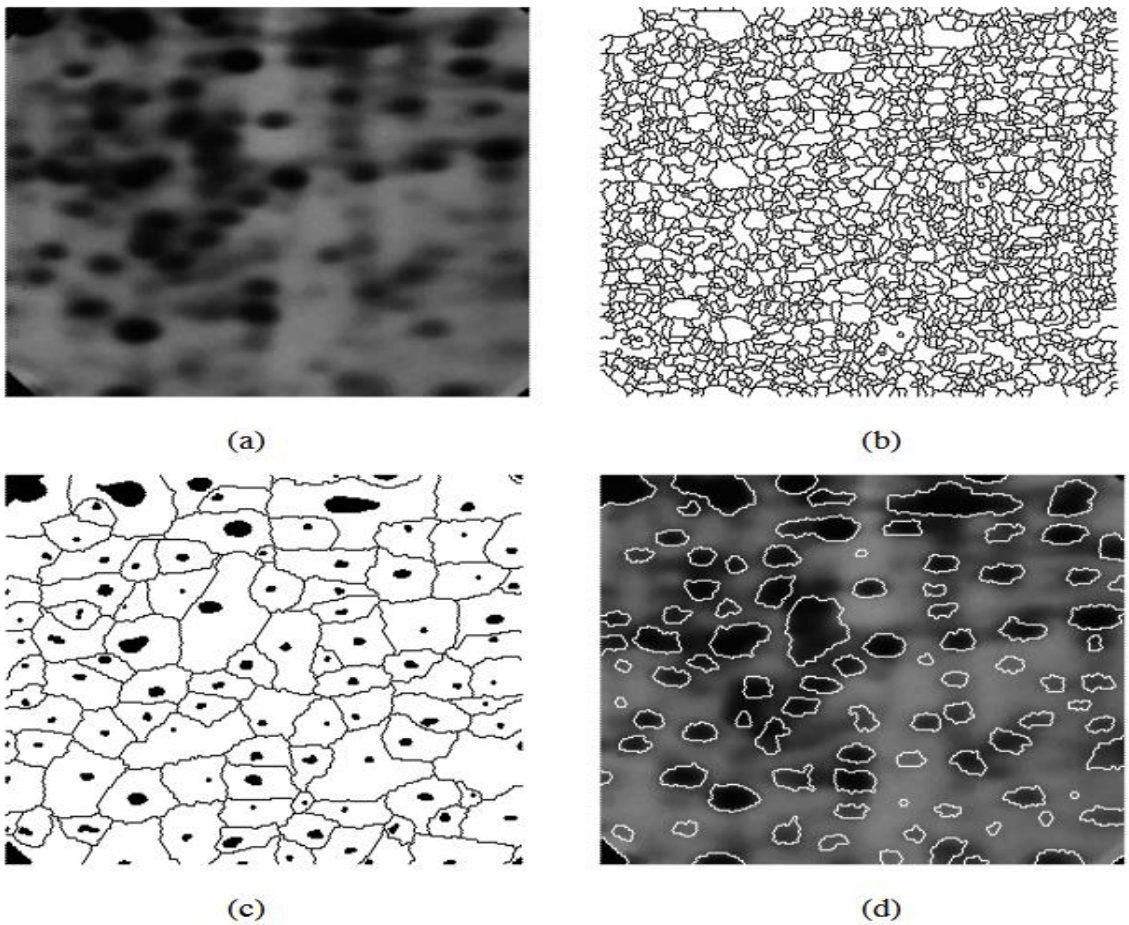


Kuva 2. Watershed -menetelmän tulvimisen eteneminen. Pystyviivat kuvaavat patoja. Eriväriset alueet kuvaavat valuma-alueita, joiden pohjalla on paikalliset minimit. [ImageJ 2018].

Watershed -menetelmän etuihin kuuluu se, että se jakaa kuvan alueisiin, jotka on eroteltu yhden pikselin paksuisilla rajoilla. Alueiden muodot mukailevat myös hyvin kuvan kohteiden muotoja. Haittoihin puolestaan kuuluu se, ettei menetelmä toimi kovin hyvin havaitsemaan ohuita rakenteita eikä alueita, joissa on pienet kontrastirajat. Suurin ongelma on kuitenkin menetelmän herkkyys kohinalle, joka johtaa *ylisegmentaatioon* (over segmentation). [Tulsani *et al*. 2013.] Ylisegmentaatiolla tarkoitetaan tilannetta, jossa segmentointi jakaa kuvan liian moneen alueeseen, jolloin kuvasta on vaikea erotella kohteita, ja segmentointi yleensä epäonnistuu. Tämä johtuu yleensä kohinasta, joka on hyvin yleistä watershed-menetelmälle [Tulsani *et al*. 2013]. Kohina muodostaa kuvaan ylimääräisiä, usein hyvin pieniä, tummempia alueita, joilla on jokaisella oma paikallinen minimi. Jokaisen näiden alueen ympärille muodostuu reunat, jolloin kuva ylisegmentoituu (kuva 3b).

Ylisegmentaation ratkaisemiseen on olemassa useita lähestymistapoja. Kohteiden laskemista varten kaikista ideaalein tapa on käyttää *markkereihin perustuvaa watershed* -menetelmää (marker based watershed) [Tulsani *et al*. 2013]. Verisolujen segmentoinnin kohdalla tämä tapa onkin varsin yleinen, koska verisolut halutaan yleensä laskea. Markkereihin perustuvassa watershed -menetelmässä määritellään aluksi osa kuvan alueista markkereiksi. Markkerit

jakaantuvat sisäisiin ja ulkoisiin markkereihin. Sisäiset markkerit kuuluvat kuvassa olevaan kohteeseen ja ulkoiset kuuluvat taustaan (kuva 3c). [Beucher and Meyer 1992]. Markkereiden valinta tapahtuu yleensä automaattisesti hyödyntäen esimerkiksi matemaattiseen morfologiaan perustuvia operaattoreita. Morfologiset operaattorit muokkaavat kuvaa sen muotojen perusteella. Sillä voi esimerkiksi muokata kuvassa olevien kohteiden kokoa, muotoa, rakennetta ja yhtenäisyyttä [Tulsani *et al*. 2013]. Niiden avulla voi esimerkiksi tasoittaa kuvan pikseleitä, jolloin kohinasta aiheutuvat minimikohdat katoavat. Tämän jälkeen markkereiksi voidaan valita tummat kohdat. Kun markkerit on valittu, suoritetaan watershed -algoritmi siten, että ainoastaan kohteeseen kuuluvia markkereita voidaan käyttää paikallisina minimeinä [Beucher and Meyer 1992]. Jokaista kuvan aluetta kohden on tällöin vain yksi sisäinen markkeri. Ylisegmentaatiota ei tapahdu, koska paikallisiksi minimeiksi ei valita ollenkaan kohinan muodostamia alueita (kuva 3d).



Kuva 3. a) Alkuperäinen kuva. b) Watershedin aiheuttama ylisegmentoituminen. c) Valitut markkerit hahmoteltuna tummina kohtina. d) Markkereihin perustuvan segmentoinnin lopputulos. [Beucher 1992.]

4.1 Käyttökohteet

Watershed -menetelmää käytetään hyvin paljon verisolujen segmentoinnissa, koska sen avulla pystytään erottelemaan päällekkäisiä soluja, mikä helpottaa niiden laskemista. Sitä yleensä käytetäänkin segmentoinnin loppuvaiheessa erottelemaan solut toisistaan.

Liu ja muut [2015] käyttivät watershed -menetelmää erottelemaan valkosolut toisistaan. He käyttivät solujen tumia markkereina, jolloin watershed -menetelmää käytettiin kahdesti. Ensimmäisellä kerralla menetelmää käytettiin erottelemaan solut, joiden tumat eivät olleet kosketuksissa muiden solujen kanssa, ja toisella kerralla erottelemaan solut joiden tumat olivat kiinni muissa soluissa.

Arslan ja muut [2014] hyödynsivät markkereihin perustuvaa watershed -menetelmää valkosolujen segmentoimisessa. He käyttävät värimuunnoksia ja etäisyyttä markkereiden määrittelyssä. Kuvasta suodatettiin kaikki määriteltyä kynnysarvoa pienemmät arvot, mikä vähensi mahdollisuutta ylisegmentaatiolle. Lopulta markkereiksi valittiin pienimmät jäljellä olevat minimikohdat.

5. Klusterointi

Klusterointi (clustering) on menetelmä, jossa datajoukko jaetaan erilaisiin ryhmiin, jotka sisältävät samankaltaisia alkioita. Sitä käytetään monilla aloilla, kuten hahmontunnistuksessa, tiedonlouhinnassa, kuvanprosessoinnissa, biologiassa, psykologiassa ja markkinoinnissa [Mohapatra *et al*. 2011].

Sovelluskentän laajuuden vuoksi erilaisia klusterointimenetelmiä on erittäin paljon, ja niiden jaottelu on vaikeaa. Yleisesti klusteroinnin voi jakaa *kovaan klusterointiin* (hard clustering) ja *pehmeään klusterointiin* (soft clustering), joka tunnetaan myös *sumeana klusterointin*a (fuzzy clustering). Yksi hyvin yleinen jaottelutapa, on jaotella menetelmät hierarkkisiin (hierarchical clustering) ja osittaviin (partitional clustering) menetelmiin. Hierarkkiset menetelmät jakavat datajoukon usean klusteroinnin avulla, ja osittavat menetelmät suorittavat jaon vain yhden klusteroinnin perusteella. [Tuononen 2005].

Klusterointia voidaan hyödyntää myös kuvan segmentoimisessa, jossa klusteroinnin avulla ryhmitellään kuvan pikseleitä perustuen pikselien arvoihin. Seuraavaksi tarkastellaan klusteroinnin toimintaa yleisemmin klusterointimenetelmän avulla.

5.1 K-means

K-means, joka tunnetaan myös Lloydin algoritmina, on tunnetuin klusterointimenetelmä. Se kuuluu osittaviin menetelmiin. Sen tavoitteena on löytää minimiarvo funktiolle:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\| x_i - v_j \| \right)^2$$

missä,

$(\| x_i - v_j \|)^2$ = euklidisen etäisyyden neliö x_i (datapisteen) ja v_j (klusterin keskikohdan) välillä.
c = klustereiden keskikohtien määrä,
c_i = datapisteiden määrä klusterissa *i.* [Naik 2010.]

K-means klusteroinnin idea on hyvin yksinkertainen. Algoritmi alkaa keskikohtien valitsemisella. Datajoukosta valitaan keskikohdat satunnaisesti. Jokainen keskikohta määrittää yhden klusterin. Tämän jälkeen toistetaan seuraavia vaiheita (kuva 4):

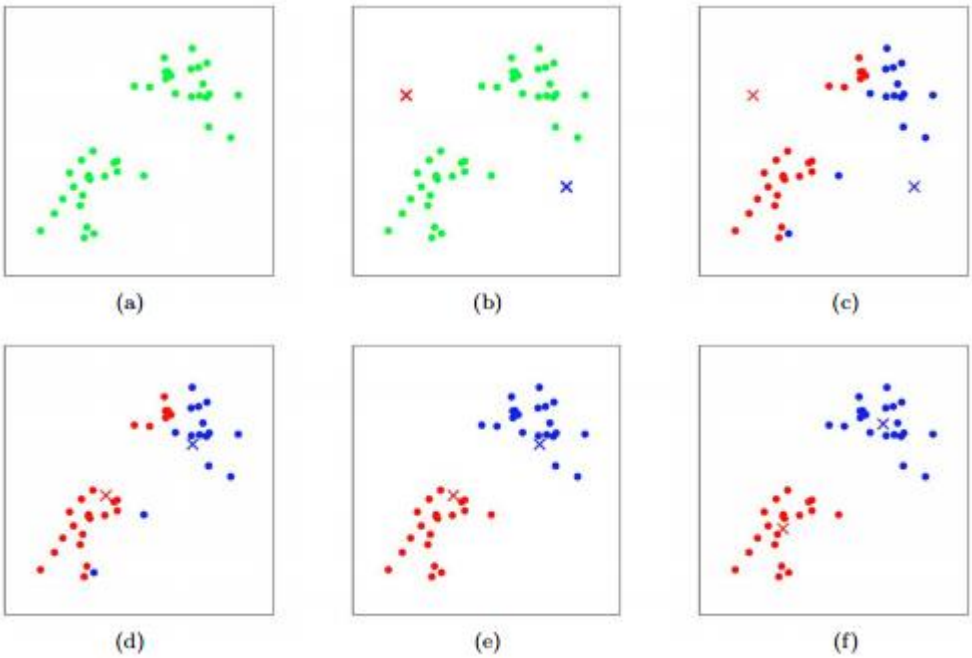
1. Lasketaan datapisteiden etäisyydet keskikohtiin ja liitetään jokainen piste lähimmän keskikohdan klusteriin.
2. Lasketaan klustereiden keskipisteet uudelleen seuraavalla kaavalla:

$$\boldsymbol{V_i = (1/c_i) \sum_{j=1}^{c_i} x_i}$$

Kun keskikohdat eivät enää muutu, ollaan päästy lopulliseen tulokseen. [Naik 2010.] K-means klusteroinnin etuihin kuuluu sen yksinkertaisuus ja tehokkuus. Se antaa parhaan tuloksen, kun datajoukot ovat erillään toisistaan. K-means klusterointiin liittyy kuitenkin paljon myös ongelmia. K-means ei pysty erottamaan päällekkäisiä klustereita toisistaan kovin hyvin. Myös kohina ja poikkeamat, sekä epälineaariset datajoukot tuottavat algoritmille ongelmia. Lisäksi algoritmi vaatii klustereiden ennalta määrittelyn, mikä vaikuttaa lopputulokseen. [Naik 2010.]

Alkuperäisten klusterien määrittely satunnaisesti vaikuttaa oleellisesti lopputulokseen. Tuloksena saattaa syntyä hyvin epäoptimaalisia klustereita. Ratkaisuna tähän ongelmaan on hyödyntää K-means++ -algoritmia, joka määrittää klusterien alkuperäiset keskikohdat perustuen niiden etäisyyksiin toisis-

taan. Algoritmi pyrkii valitsemaan keskikohdat siten, että ne olisivat kaukana muista keskikohdista. Ensimmäinen keskikohta valitaan satunnaisesti. Tämän jälkeen muut keskikohdat valitaan etäisyyksien mukaan. Mitä kauempana piste on lähimmästä valitusta keskikohdasta, sitä isommat mahdollisuudet sillä on tulla valituksi uuden klusterin keskikohdaksi. Kun keskikohdat on valittu, K-means klusterointi suoritetaan normaalisti. K-means++ eroaa siis tavallisesta K-means klusteroinnista pelkästään keskikohtien valitsemisessa. Se tuottaa kuitenkin paremman tarkkuuden ja on nopeampi. [Arthur and Vassilvitskii 2007.]



Kuva 4. K-means klusteroinnin vaiheet havainnollistettuna. a) Alkutilanne. Vihreitä datapisteitä ei ole jaettu klustereihin. b) Valitaan klustereiden keskikohdat satunnaisesti. Merkataan niitä erivärisillä merkeillä. c) Datapisteet jaetaan lähimpiin klustereihin. d) Lasketaan klustereiden keskipisteet uudelleen. e) Jaetaan datapisteet klustereihin uudelleen. f) Lopullinen tulos, johon päästään, kun vaiheita c-e on iteroitu niin kauan, kunnes klustereiden keskikohta ei enää muutu. [Piech 2013]

5.2 Käyttökohteet

Klusterointia käytetään hyvin paljon kuvan segmentoinnissa. Segmentoinnissa datajoukkona toimii kuvan pikselit, jotka jaetaan erilaisiin klustereihin perustuen pikselien arvoihin. Aiemmin esitelty K-means klusterointi on yksi suosituimmista klusterointimenetelmistä myös segmentoinnissa.

Mohapatra ja muut [2011] sovelsivat *karkeita joukkoja* (rough sets) K-means klusterointiin, ja hyödynsivät sitä valkosolujen segmentoinnissa. Rough K-means klusteroinnissa jokaisella klusterilla on ylä- ja ala-arvot, joihin datapisteet luokitellaan. Ala-arvoihin kuuluu ne datapisteet, jotka kuuluvat varmasti

klusteriin. Yläarvoihin puolestaan kuuluvat pisteet, jotka voivat mahdollisesti kuulua klusteriin. Tavalliseen K-means klusterointiin verrattuna kaikkia pisteitä ei siis määritetä varmasti kuuluvaksi johonkin tiettyyn klusteriin, mikä toimii yhtenä ratkaisuna päällekkäisten datapisteiden tuottamaan ongelmaan.

Zhang ja muut [2014] käyttivät K-means klusterointia valkosolujen sytoplasman ja tuman segmentoinnissa. He hyödynsivät RGB-, HSI- ja CMYK -värimalleja, joiden eri kanavien avulla voidaan nähdä kuvan verisolut eri tavalla. Esimerkiksi RGB -värimallin sinisellä kanavalla ja CMYK -värimallin keltaisella kanavalla punasolut ja valkosolujen tumat sisältävät isomman kontrastin kuin valkosolujen sytoplasma. Eri värimallien avulla verisolut segmentoitiin käyttäen K-means klusterointia, ja tulokseksi saatiin 94.6% ja 95.1% tarkkuudet riippuen parametrista. Verisolujen segmentoinnissa tarkkuudet lasketaan yleensä vertailemalla manuaalisesti laskettuja tuloksia menetelmän tuottamiin tuloksiin.

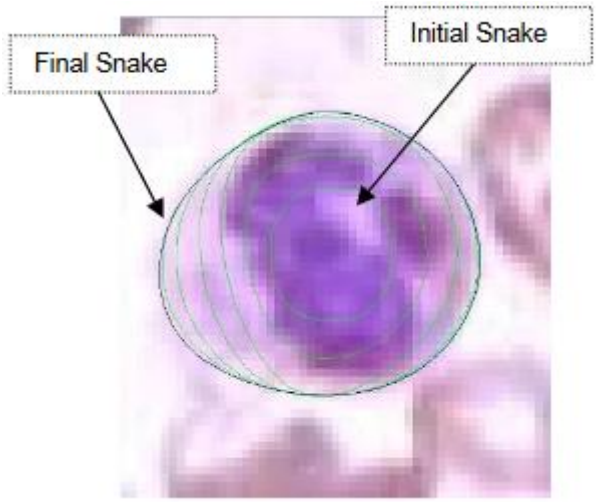
K-means klusteroinnin lisäksi myös muita klusterointimenetelmiä on käytetty verisolujen segmentoimisessa. Liu ja muut [2015] käyttivät mean shift klusterointia valkosolujen segmentoimisessa. Mean shift klusteroinnissa klusterit muodotetaan perustuen tiheyteen ja pisteiden etäisyyteen. Klusterit muodostuvat kohtiin, jossa tiheys on korkea, ja klusterien ympärillä olevat pikselit liikkuvat kohti lähintä klusteria. Theera-Umpon [2005] käytti fuzzy-C means klusterointia valkosolujen tuman ja sytoplasman segmentoimisessa. Fuzzy-C -means on sumea versio perinteisestä K-means klusteroinnista. Erona on se, että datapisteillä on paino, joka määrittää, kuinka vahvasti ne kuuluvat eri klustereihin. Datapisteet voivat siis kuulua samaan aikaan useampaan eri klusteriin.

6. Muut menetelmät

Verisolujen segmentoimisessa on käytetty myös muita segmentointimenetelmiä. *Luokittimien* käyttö on nykyään hyvin yleistä segmentoinnissa, ja ne soveltuvat hyvin myös verisolujen segmentointiin. Luokittimien käyttö muistuttaa hyvin paljon klusterointia. Erona on se, että luokittimet perustuvat ohjattuun oppimiseen, kun taas klusterointi perustuu ohjaamattomaan oppimiseen. Luokittimissa käytetään aiempaa tietämystä, jolla luokat voidaan määritellä. Sen sijaan klusteroinnissa pyritään löytämään yhtäläisyyksiä datan välillä, ja jakamaan data sen perusteella. Yleisimpiä luokittimia, joita verisolujen segmentoimisessa on käytetty, on mm. tukivektorikone (support vector machine), neuroverkot, naiivi Bayes -luokittelija ja K:n lähimmän naapurin menetelmä. Luokittimet on yleisesti todistettu tehokkaiksi, mutta ne eivät välttämättä toimi hyvin, mikäli erot opetusjoukkojen ja testijoukon välillä on suuria esimerkiksi kuvan-

tamisolosuhteiden seurauksena. Lisäksi myös opetukseen vaadittavien näytteiden saatavuus voi olla ongelma. [Zheng *et al.* 2018.]

Luokittimien lisäksi toinen verisolujen segmentoimisessa tehokkaaksi todistettu menetelmä on *aktiiviset reunamallit* (active contour models), jotka tunnetaan myös *käärmeinä* (snakes). Nimensä mukaisesti menetelmässä hyödynnetään käärmeitä muistuttavia muotoaan muuttavia käyriä, jotka asetetaan aluksi segmentoitavien kohteiden lähelle. Käärme liikkuu kohti kohteen reunoja perustuen laskettuihin ulkoisiin ja sisäisiin voimiin. Ulkoiset voimat työntävät käärmettä kohti kohteen reunoja ja sisäiset voimat pitävät käärmeen yhtenäisenä. Lopulta käärme asettuu kohteen ääriviivoihin kiinni (kuva 5). [Hamghalam *et al.* 2009.] Malleja on useita erilaisia, mutta yleisin niistä on gradient vector flow (GVF) -käärme, jota esimerkiksi Hamghalam ja muut [2009] käyttivät. Aktiivisten reunamallien avulla voidaan siis löytää solujen reunat. Päällekkäiset solut tuottavat menetelmälle kuitenkin ongelmia, ja tulokset riippuvat paljolti alkuperäisestä käärmeestä [Zheng *et al.* 2018].



Kuva 5. Alkuperäinen käärme on kuvassa sisimpänä. Lopullinen käärme on asettunut solun ympärille. [Hamghalam *et al.* 2009]

7. Yhteenveto

Kuvien segmentointi on vaikea tehtävä. Ei ole olemassa yhtä ratkaisua, joka toimii kaikissa tilanteissa. Erilaisiin sovelluksiin tarvitaan erilaisia segmentointimenetelmiä, joiden tehokkuus riippuu sovelluksen ominaisuuksista. Tässä tutkielmassa tarkasteltiin verisolujen segmentointia, johon liittyy paljon erilaisia ongelmia. Verisolujen segmentoinnissa täytyy huomioida erityisesti solujen päällekkäisyyksien aiheuttamat ongelmat, mikä on tärkeää, koska solut halutaan usein erottaa toisistaan, jotta ne pystytään laskemaan ja tunnistamaan paremmin.

Tutkielmassa käsiteltiin useita verisolujen segmentoinnissa käytettyjä menetelmiä. Menetelmät eroavat toisistaan hyvin paljon. Kaikilla niillä on jokin tehtävä, jossa ne toimivat erityisen hyvin. Esimerkiksi watershed -menetelmän avulla verisolut voidaan erotella toisistaan, kun taas kynnystyksen avulla voidaan tuottaa binäärikuvia verisoluista. Tutkielmassa käsiteltiin kuitenkin vain joitain yleisimpiä menetelmiä. Näiden lisäksi on olemassa laaja kirjo muita menetelmiä, joita verisolujen segmentoimisessa käytetään. Nykyisten menetelmien lisäksi myös uusia menetelmiä tarvitaan lisää, mikä on edellytys sille, että segmentoinnista saadaan yhä tarkempaa ja tehokkaampaa.

Viitteluettelo

Salim Arslan, Emel Ozyurek, and Cigdem Gunduz- Demir. A color and shape based algorithm for segmentation of white blood cells in peripheral blood and bone marrow images. *Cytometry Part A* 85, 6, 480-490.

David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In: *Proc. of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 1027-1035.

Ewert Bengtsson. 2003. Computerized cell image analysis: past, present, and future. In: *Proc. of the Scandinavian Conference on Image Analysis*, 395-407.

Serge Beucher and Fernand Meyer. 1992. The morphological approach to segmentation: the watershed transformation. *Optical Engineering-New York-Marcel Dekker Incorporated* 34, 433-433.

Serge Beucher. 1992. The watershed transformation applied to image segmentation. *Scanning Microscopy Supplement 6,* 299-314.

Salam Devi, Singha Joyeeta, Sharma Manish, and Hussain Rabul. 2017. Erythrocyte segmentation for quantification in microscopic images of thin blood smears. *Journal of Intelligent & Fuzzy Systems* 32, 4, 2847-2856.

Leyza Dorini, Minetto Rodrigo, and Jeronimo Neucimar. 2013. Semiautomatic white blood cell segmentation based on multiscale analysis. *IEEE Journal of Biomedical and Health Informatics* 17, 1, 250-256.

Mohammad Hamghalam, Mohammad Motameni, and Aghil Esmaeili Kelishomi. 2009. Leukocyte segmentation in giemsa-stained image of peripheral blood smears based on active contour. In: *Proc. of the 2009 International Conference on Signal Processing Systems*, 103-106.

Der-Chen Huang, Kun-Ding Hung and Yung-Kuan Chan. 2012. A computer assisted method for leukocyte nucleus segmentation and recognition in blood smear images. *Journal of Systems and Software* 85, 9, 2104-2118.

ImageJ. 2018. Classic Watershed. https://imagej.net/Classic_Watershed. Checked 30.11.2018.

Zhi Liu, Liu Jing, Xiao Xiaoyan, Yuan Hui, Li Xiaomei, Chang Jun and Zheng Chengyun. 2015. Segmentation of white blood cells through nucleus mark watershed operations and mean shift clustering. *Sensors* 15, 9, 22561-22586.

Subrajeet Mohapatra, Dipti Patra, and Kundan Kumar. 2011. Blood microscopic image segmentation using rough sets. In: *Proc. of the Image Information Processing (ICIIP), 2011 International Conference*, 1-6.

Bryan Morse. 2000. Lecture 4: Thresholding. Brigham Young University. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf. Checked 4.10.2018.

Azad Naik. 2010. K-means clustering algorithm. https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm. Checked 7.11.2018.

Nobuyuki Otsu. 1979. A threshold selection method from gray-level histogram, *IEEE Transactions Systems Man, and Cybernetics* 9, 62-66.

Chris Piech. 2013. K Means. http://stanford.edu/~cpiech/cs221/handouts/kmeans.html Checked 30.11.2018

Ravindra Sarode. 2018. Components of Blood. https://www.msdmanuals.com/home/blood-disorders/biology-of-blood/components-of-blood. Checked 20.10.2018.

Linda G. Shapiro and George C. Stockman. 2001. *Computer Vision*, 305-350.

Mehmet Sezgin and Bülent Sankur. 2004. Survey over image thresholding techniques and quantitative performance evaluation, *Journal of Electronic imaging* 13, 146-165.

Nipon Theera-Umpon. 2005. White blood cell segmentation and classification in microscopic bone marrow images. In: *Proc. of the International Conference on Fuzzy Systems and Knowledge Discovery*, 787-796.

Marko Tuononen. 2005. Klusterointimenetelmät. http://cs.joensuu.fi/~mtuonon/Klusterointimenetelmat.pdf. Katsottu 6.11.2018.

Hemant Tulsani, Saransh Saxena and Naveen Yadav. 2013. Segmentation using morphological watershed transformation for counting blood cells. *IJCAIT* 2, 3, 28-36.

Congcong Zhang, Xiao Xiaoyan, Li Xiaomei, Chen Ying-Jie, Zhen Wu, Chang Jun, Zheng Chengyun, and Liu Zhi. 2014. White blood cell segmentation by color-space-based k-means clustering. *Sensors* 14, 9, 16128-16147.

Xin Zheng, Wang Yong, Wang Guoyou, and Liu Jianguo. 2018. Fast and robust segmentation of white blood cell images by self-supervised learning. *Micron* 107, 55-71.

Tiedostojen jakaminen vertaisverkoissa – BitTorrent, IPFS ja Dat

Laura Ketola

Tiivistelmä.
Tutkielma käy läpi vertaisverkkojen taustaa ja toteutusta, keskittyen erityisesti vertailemaan kolmea eri vertaisverkkoteknologiaa keskenään. BitTorrent on näistä käytetyin vertaisverkkoprotokolla. IPFS ja Dat puolestaan ovat uudempia teknologioita, joilla on mielenkiintoisia mahdollisia käyttökohteita.

Avainsanat ja -sanonnat: P2P, tiedostojen jakaminen, vertaisverkot.

1. Johdanto

Tässä tutkielmassa teen katsauksen erilaisiin tiedostojen jakoon suunnattujen vertaisverkkoprotokollien toimintaan, niiden käyttämiseen ja niistä kirjoitettuun kirjallisuuteen. Lisäksi vertailen valittuja teknologioita keskenään.

Vertaisverkot ovat hajautettujen järjestelmien alakategoria, jossa verkkoa käyttävät sovellukset välittävät tietoa keskenään. Vertaisverkkoihin kuuluu esimerkiksi useiden median suoratoistopalvelujen toteutukset ja kryptovaluutat. Tässä tutkielmassa keskityn kuitenkin vertaisten väliseen tiedostonjakoon soveltuviin vertaisverkkoihin. Vertailuun on valittu BitTorrent [Cohen 2003], IPFS [Benet 2014] ja Dat [Robinson *et al.* 2018]. Nämä kaikki soveltuvat tiedostojen jakoon vertaisten välillä, mutta ovat kukin erikoistuneet hiukan erilaisiin käyttötarkoituksiin. Valituista protokollista BitTorrent on nykyisin käytetyin vertaisverkkoprotokolla ja kaksi jälkimmäistä – IPFS ja Dat – puolestaan ovat melko uusia teknologioita.

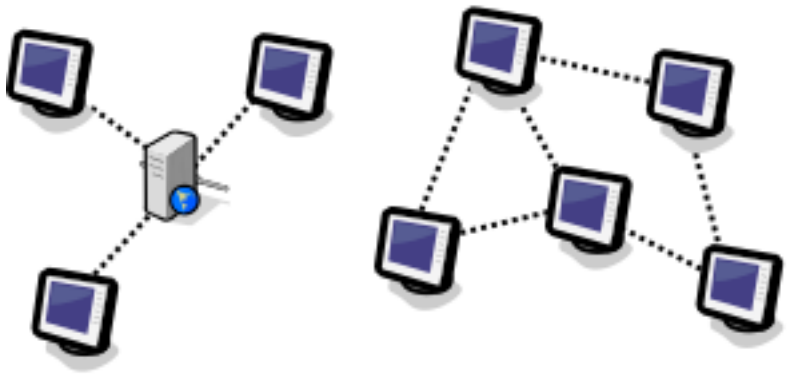
Vertaisverkoista on kirjoitettu paljon, mutta omien selvitysteni mukaan suomeksi julkaistua tutkimusta on verrattain vähän, ja se keskittyy pääosin vertaisverkkojen uhkiin ja laittomiin käyttötarkoituksiin. Tämän tutkielman yhtenä päämääränä on siis myös koota ja järjestellä englanniksi kirjoitettua materiaalia yhteen suomeksi.

Luvussa 2 tutustun vertaisverkkoteknologioihin yleisesti, avaten niihin liittyvää käsitteistöä ja taustaa. Luvuissa 3, 4 ja 5 tutustun tarkemmin valittuihin vertaisverkkoprotokolliin. Luvussa 6 tarkastelen näiden eroja ja samankaltaisuuksia. Luku 7 on yhteenveto.

2. Vertaisverkoista

Tyypilliset internet-sivustot toimivat asiakas-palvelin-mallilla. Tässä mallissa *asiakkaana* toimiva tietokone pyytää tietoverkon yli tiedostoa *palvelimena* toimivalta tietokoneelta, ja palvelin vastaa palauttamalla pyydetyn tiedoston. Tietoliikenne siis tapahtuu asiakkaan ja palvelimen välillä. [Kattelus 2008]

Vertaisverkkoarkkitehtuurit sen sijaan häivyttävät asiakkaiden ja palvelimien eron, sillä verkossa toimivat laitteet jakavat tiedostoja suoraan toisilleen ja toimivat kukin vuorollaan asiakkaana ja palvelimena. Näitä ohjelmia, jotka jakavat toisilleen tietoa, kutsutaan *vertaisiksi* (peers). Vertaisten muodostamaa joukkoa kutsutaan edelleen *parveksi* (swarm). Kuvassa 1 on verrattu asiakas-palvelin-arkkitehtuuria ja vertaisverkkoarkkitehtuuria. [Androutsellis-Theotokis *et al.* 2014; Cohen 2003]



Kuva 1. Vasemmalla on asiakas-palvelin-arkkitehtuuri, ja oikeanpuolinen kuva esittää vertaisverkkoa. [Androutsellis-Theotokis *et al.* 2014]

Koska tiedon siirto voi käyttää tasaisesti kaikkien vertaisten tietoliikennekapasiteettia, kyetään sekä teoriassa että käytännössä hyödyntämään todellinen kapasiteetti tehokkaammin kuin palvelin-asiakas -arkkitehtuurilla [Cohen 2003]. Vertaisverkot myös kestävät asiakas-palvelin -verkkoja paremmin verkon rakenteen muutoksia. Tällaisia verkon rakenteen muutoksia tapahtuu esimerkiksi kun vertaiset liittyvät ja poistuvat verkosta nopeaan tahtiin [Androutsellis-Theotokis *et al.* 2014]. Vuosina 2002–2007 vertaisverkkoliikennettä oli yli puolet kaikesta Internetin liikenteestä, kunnes median suoratoistopalvelut alkoivat yleistyä [Varvello *et al.* 2012].

Vertaisten muodostamasta verkosta käytetään käsitettä *päällysverkko* (overlay network), joka yhdistää vertaiset johonkin rakenteeseen, joka on riippumaton tavallisen Internet-verkon rakenteesta. Vertaisverkkojen keskeisiin teknisiin haasteisiin kuuluu tämän päällysverkon rakentaminen eli käytännössä muiden vertaisten löytäminen. [Androutsellis-Theotokis *et al.* 2014]

Vertaisten löytämisen ongelman ratkaisuista yleisin ja teknisesti yksinkertaisin on joukko keskuspalvelimia, jotka pitävät listaa vertaisista ja välittävät niiden IP-osoitteita eteenpäin. Tällaisessa konfiguraatiossa nämä keskuspalvelimet muodostavat yksittäisen kriittisen pisteen, mikä vähentää järjestelmän luotettavuutta. Data ei kuitenkaan kulje tämän seurantapalvelimen kautta, vaan se vain tuo vertaiset yhteen. [Kattelus 2008]

Toinen nykyisin yleinen vertaisten löytämisen mekanismi on *hajautettu tiivistetaulu* tai akronyymiä käyttäen *DHT* (distributed hash table). Nämä ovat rakenteeltaan samankaltaisia kuin tavalliset hajautustaulut, eli ne yhdistävät *avaimen* (key) tiettyyn *arvoon* (value). Hajautetut tiivistetaulut kuitenkin jakavat tiedon useam-

malle vertaiselle siten, että jos vertainen tietää yhden toisen vertaisen, se kykenee hakemaan minkä tahansa arvon verkosta. Usein DHT-verkkoja käytetään yhdistämään etsittävän tiedoston tiiviste joukkoon vertaisia, joilla on tiedoston osia jaettavana. Näin käytettynä vertaisverkko ei ole riippuvainen keskitetystä seurantapalvelimesta. [Maymounkov *et al.* 2002]

Vaikka erot riippuvat toteutuksesta, tavallisesti seurantapalvelimet ovat tehokkaampi tapa tuoda vertaiset yhteen. Tämä kuitenkin tekee seurantapalvelimesta potentiaalisen pullonkaulan ja sen rikkoutuminen käytännössä estää koko verkon toiminnan. [Cohen 2003]

Koska vertaisverkoissa vertaiset jakavat omaa tiedonsiirto- ja tallennuskapasiteettiaan, siihen osallistuminen ei ole ilmaista vertaiselle. Yksi vertaisverkkojen haasteista liittyy resurssien jakamiseen eli siihen, miksi vertainen ei vain hyväksikäyttäisi verkkoa, jolloin hän pyytäisi dataa mutta ei kuitenkaan jakaisi sitä eteenpäin. Tällaisia vertaisia, jotka vain lataavat tietoa eivätkä jaa sitä, kutsutaan *loisiksi* (leecher). Tämän ongelman vuoksi useimmat vertaisverkot käyttävät mekanismeja, jotka pyrkivät suosimaan vertaisia jotka lataavat ja jakavat tietoa samassa suhteessa. [Cohen 2003; Benet 2014]

Vertaisverkkoihin liittyy myös vahvasti ongelma yksityisyydestä. Jotta vertainen voi ladata parvelta dataa, muiden vertaisten on oltava tietoinen sen verkko-osoitteesta ja mitä se on lataamassa. Käyttäjät eivät aina ole tietoisia heikosta yksityisyyden tasosta. [Androutsellis-Theotokis *et al.* 2014]

Vertaisverkoissa jaettu materiaali on myös usein luonteeltaan laitonta tai vaarallista. Vertaisverkkojen luonteen vuoksi niiden sensurointi tai rajoittaminen on kuitenkin vaikeaa tai jopa mahdotonta, sillä jokainen vertainen voi itsenäisesti päättää mitä se lataa ja jakaa. Tämän vuoksi mikään keskeinen taho ei voi estää tietyn tiedoston levittämistä. [Watters *et al.* 2011]

Vertaisverkot voidaan luokitella esimerkiksi niiden keskittyneisyyden tai rakenteen perusteella. Ensimmäiseksi sukupolveksi voidaan kutsua hyvin keskitettyjä vertaisverkkoja, joihin kuului esimerkiksi Napster. Vastaavasti toiseksi sukupolveksi voidaan käsittää vahvemmin hajautetut vertaisverkot, joista ensimmäisiä oli Gnutella. [Androutsellis-Theotokis *et al.* 2014]

Rakenteesta puhuttaessa vertaisverkot jaetaan usein kahteen luokkaan: *järjestettyihin* (structured) ja *järjestämättömiin* (unstructured). Järjestämättömissä verkoissa vertaiset eivät muodosta tiettyä rakennetta, vaan jaettava tieto on erilaisin algoritmein etsittävä parvesta. Järjestetyt verkot pitävät yllä vertaisten keskinäistä rakennetta, jolloin etsittävä tieto on löydettävissä helpommin. Tällainen rakenne on kuitenkin vaikea ylläpitää. [Androutsellis-Theotokis *et al.* 2014]

2.1. Aiheeseen liittyvää käsitteistöä

Kryptografinen tiiviste, tai pelkkä *tiiviste* on matemaattinen funktio, joka tiivistää mielivaltaisen pituisen tiedoston ennakoitavasti tietyn mittaiseksi merkkijonoksi.

Hyvän tiivisteen ominaisuuksiin kuuluu olla mahdollisimman uniikki ja olla mahdoton palauttaa takaisin alkuperäiseksi tiedostoksi. Jos kahdella tiedostolla on sama tiiviste, voidaan olla käytännössä varmoja, että kyseessä on sama tiedosto. Näin ollen tiivisteitä voidaan käyttää tiedoston sisällön varmentamiseen.

Julkisen avaimen salaus (public key cryptography) on kryptografinen salausmenetelmä, jossa käytetään kahta erillistä avainta: *yksityistä avainta* (private key) ja *julkista avainta* (public key). Salaamalla tieto käyttäen julkista avainta, vain vastaavan yksityisen avaimen haltija kykenee purkamaan salauksen. Vaihtoehtoisesti voidaan myös allekirjoittaa tiedosto käyttäen yksityistä avainta, jolloin julkisella avaimella voidaan varmistaa että tiedoston on allekirjoittanut yksityisen avaimen haltija.

DNS (Domain Name System) on Internetin nimipalvelinjärjestelmä joka yhdistää domain-nimen (kuten uta.fi) Internet-osoitteeseen (kuten 153.1.6.41).

3. BitTorrent

BitTorrent [Cohen 2003] on käytetyin ja tunnetuin vertaisverkkoprotokolla [Varvello *et al.* 2012]. Se on myös vanhin tässä tutkielmassa käsitellyistä vertaisverkoista. Se on erityisen tunnettu sen käytöstä tekijänoikeuksia ja rikkovan materiaalin levittämiseen. Selkeästi suurin osa BitTorrentilla jaettavasta materiaalista on laitonta [Watters *et al.* 2011]. Sillä on myös merkittävä historiallinen rooli vertaisverkkoteknologioiden kehityksessä.

Tekijänoikeuksia rikkovan tai muuten laittoman median lisäksi BitTorrent soveltuu myös muiden suurien tiedostojen tehokkaaseen levittämiseen. Akateemisesti kiinnostava käyttökohde on esimerkiksi Academic Torrents -projekti [Cohen *et al.* 2014] joka organisoi tieteellisten artikkeleiden ja datajoukkojen levittämistä BitTorrent-verkon avulla.

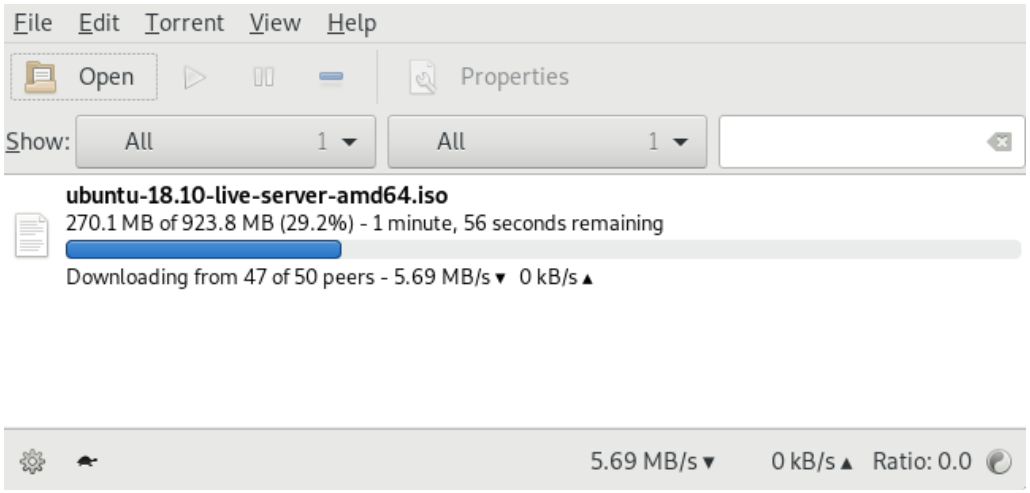
BitTorrent lasketaan teknisesti ensimmäisen sukupolven vertaisverkoksi, sillä sen seurantapalvelimet ovat keskitettyjä. Toisaalta BitTorrent tukee myös DHT:n käyttämistä, mikä tekee siitä sekä toisen sukupolven verkon että järjestetyn verkon.

3.1. Käyttäminen

Jakaakseen tiedostoja BitTorrentilla, alkuperäinen jakaja tarvitsee BitTorrent-sovelluksen, seurantapalvelimen joka pitää kirjaa tietoa jakavan parven vertaisista, *Torrent-metatietotiedoston* joka sisältää esimerkiksi seurantapalvelimen osoitteen, ja WWW-palvelimen jolla metatiedostoa voi levittää. [Cohen 2003]

Kun lataaja haluaa ladata jaetut tiedostot, hän tarvitsee sekä BitTorrent-sovelluksen että tavalliselta HTTP-palvelimelta lataamansa Torrent-tiedoston. Torrent-tiedostoa käyttäen sovellus ottaa yhteyden parveen ja lataa siltä tarvittavat tiedostot. Haettuaan kaiken tarvitsemansa tiedon, vertainen siirtyy puolestaan jakamaan tietoa, joka hänellä on, kunnes asiakasohjelma suljetaan. Kuvassa 2 käyttäjä on lataamassa

Ubuntu-käyttöjärjestelmän levykuvaa BitTorrent-parvelta. [Cohen 2003]



Kuva 2. Kuvankaappaus "Transmission" BitTorrent sovelluksesta lataamassa tiedostoa. [Transmission Project 2005]

Vaihtoehtoinen tapa päästä käsiksi BitTorrent-parven tiedostoihin ovat *magneettilinkit* (magnet link) joihin on koodattu metatiedostojen sisältö. Näin ei ole tarpeen ladata erillistä Torrent-tiedostoa, vaan magneettilinkin voi avata suoraan BitTorrent-sovelluksessa. [Hazel and Norberg 2008]

3.2. Tekninen kuvaus

Kuten edelle on mainittu, vertaiset aloittavat latauksen hakemalla listan sopivista muista vertaisista seurantapalvelimelta, jonka osoite on saatu Torrentin metatietojen mukana [Cohen 2003]. Vaihtoehtoisesti seuranpalvelimen sijaan, voidaan vertaiset löytää myös käyttäen Kademlia-pohjaista [Maymounkov *et al.* 2002] hajautettua tiivistetaulua, jolloin seurantapalvelinta ei tarvita [Loewenstern and Norberg 2008].

Ladattaessa tiedostoa, se jaetaan *palasiin* (piece), tyypillisesti 256 kilotavun kokoisiin osiin. Sovellukset pyrkivät löytävät itselleen optimaalisen järjestyksen ladata näitä palasia. Tyypillisesti osia ladattaessa parvelta priorisoidaan harvinaisimpia paloja. Näin lataajalla on itsellä jotain hyödyllistä jaettavaa mahdollisimman pian, minkä lisäksi yleisemmät palaset on saatavissa todennäköisesti myöhemminkin. Tähän järjestykseen on kaksi poikkeusta: latauksen alkaessa valitaan ensimmäinen palanen satunnaisesti ja latauksen ollessa lähes valmis muutamia viimeisiä paloja pyydetään kaikilta vertaisilta, jotta ei jäädä jumiin hitaasti latautuvaan viimeiseen palaseen. Nämä ovat tosin vain tyypillisiä osia algoritmeista, protokolla itsessään jättää osien vaihtamisen suhteellisen avoimiksi. Tiettyä tiedostoa lataavat ja jakavat vertaiset muodostavat oman verkkonsa, joka ei ole yhteydessä muihin BitTorrentia käyttäviin vertaisiin. [Cohen 2003]

4. IPFS

IPFS (InterPlanetary File System) [Benet 2014] on kokoelma matalan tason vertaisverkkoteknologioita, jonka tavoite on yhdistää kaikki tiedostot yhdeksi yleiseksi tiedostojärjestelmäksi. IPFS-protokollassa tiedostoihin viitataan niiden sisällön perusteella, eli käytännössä niiden tiivisteellä. Tämän vuoksi linkit tiedostoihin IPFS:ssä viittaavat aina turvallisesti tiettyyn versioon tiedostosta, joten ei ole merkitystä kuka tiedostoa konkreettisesti jakaa. Benetin [2014] mukaan näin voisi syntyä luotettavampi ja kestävämpi vaihtoehto nykyiselle Internetille.

4.1. Käyttäminen

IPFS vaatii toimiakseen asiakasohjelman, jolla käyttäjä pystyy lataamaan parvelta tiedostoja. Tiedostoihin viitataan osoitteilla, jotka ovat muotoa */ipfs/(tiiviste)*. Esimerkiksi tässä tutkielmassa lähteenä käytetty Benetin [2014] artikkeli on haettavissa osoitteesta */ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps*. Nykyisen Internetin kanssa yhteensopivuuden vuoksi on olemassa myös *HTTP-väyliä* (HTTP gateway), jotka tarjoavat pääsyn IPFS-verkkoon ilman erillistä sovellusta. Edellä mainitun tiedoston voi avata myös verkko-osoitteesta `https://ipfs.io/ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps`. [Benet 2014]

Jos käyttäjä haluaa jakaa tiedoston, hän lisää sen IPFS-sovelluksella vertaisverkkoon, jolloin siitä muodostetaan IPFS-objekti, jonka lopullinen tiiviste palautetaan käyttäjälle ja julkaistaan IPFS-verkkoon. Tämän jälkeen tätä tiivistettä linkkinä käyttäen toinen käyttäjä kykenee hakemaan tiedoston IPFS-verkosta. Kun tämä toinen käyttäjä on hakenut itselleen tiedoston, myös hän kykenee jakamaan sitä eteenpäin. [Benet 2014]

Koska IPFS-protokollassa tiedostoihin viitataan niiden tiivisteellä, linkit niihin viittaavat aina tiettyyn versioon tiedostosta. Tällä on tilanteesta riippuen sekä etuja että haittoja. Kun voidaan olla varmoja tiedoston sisällöstä, ei tarvitse huolehtia linkkien sisällön muuttumisesta tai tiedon katoamisesta. Tästä seuraa, ettei myöskään ole tarpeen kertoa milloin jokin Internet-lähde on haettu kun siihen viitataan tekstissä lähteenä. [Benet 2014]

Tietyissä tilanteissa kuitenkin tarvitaan muuttuvaan sisältöön linkittämistä. Tämän mahdollistamiseksi IPFS:llä on nimiavaruus *IPNS* (InterPlanetary Name Space). Käyttäjä voi yksityisellä avaimellaan digitaalisesti allekirjoittaa ilmoituksen oman julkisen avaimensa määrittämän tilan uusimman version tiivisteestä. Toisena vaihtoehtona on käyttää olemassa olevaa DNS-järjestelmää muodostamaan linkki DNS-merkinnän ja IPFS-linkin välille. Näin DNS-merkintää päivittämällä voidaan vaihtaa IPNS-osoitteen sisältöä. Esimerkiksi IPFS-osoite `ipns/ipfs.io` voi viitata DNS-merkinnän mukaisesti tiettyyn IPFS-tiedostoon. [Benet 2014]

Koska IPFS antaa vahvoja takuita tiedostojen muuttumattomuudesta, kopioimattomuudesta ja säilyvyydestä, sitä on sovellettu esimerkiksi tiedostojen arkis-

tointiin. Alam ja muut [2016] kehittivät järjestelmän Internetin arkistointiin käytettävien WARC-tiedostojen arkistointiin IPFS:ssä. [Benet 2014]

4.2. Tekninen kuvaus

IPFS järjestää tiedon *merklepuuhun* (merkle tree tai hash tree). Merklepuu on suunnattu syklitön graafi, jonka jokainen solmu sisältää tiivisteen sen lapsisolmusta tai lehden tapauksessa sen sisältämästä datasta. Tästä seuraa, että solmun tiiviste on samalla myös kaikkien sen lasten identiteetti. Koodikatkelmassa 1 on esitetty IPFS:n tiedostoformaatti mukaillen Benetin esittämää formaattia. Puun käsittelyyn ja säilyttämiseen käytetään Googlen kehittämää *protobuf*-formaattia. [Benet 2014]

```
class IPFSObjekti {
    // Lista linkeistä,
    // Usein käytännössä hakemistossa
    // sijaitsevat tiedostot.
    List<IPFSLinkki> linkit;

    // Tiedoston sisältö.
    // Voi olla myös tyhjä,
    // jos kyseessä on hakemisto
    List<byte>       data;
}

class IPFSLinkki {
    String     nimi;

    // Kohteen tiiviste.
    // Multihash-formaatissa
    // tiivistefunktio on enkoodattu
    // dataan.
    Multihash hash;

    // Kohteen koko tavuina
    int       koko;
}
```

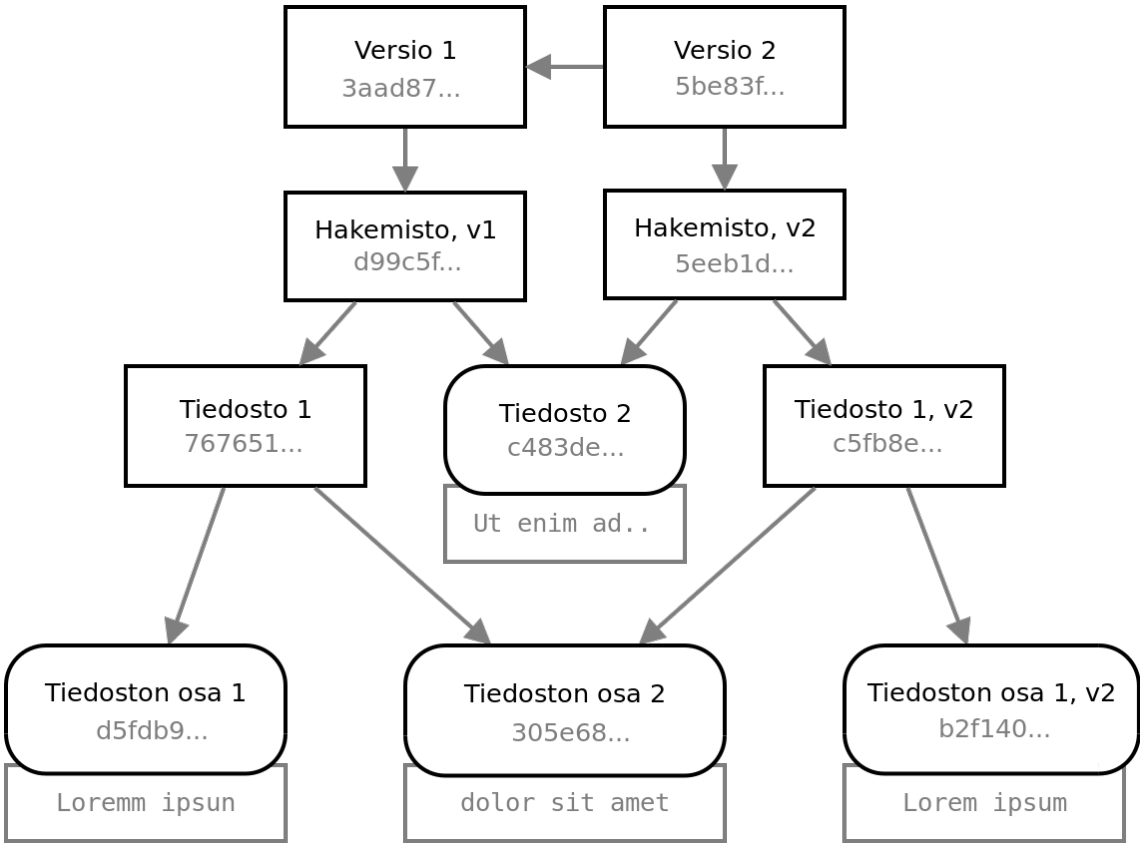
Koodikatkelma 1. IPFS merklepuu

Tämän merklepuun kuvaama tieto on kunkin sovelluksen tulkittavissa, mutta viitetoteutus tarjoaa tavan esittää tavalliset hakemistot ja tiedostot merklepuuta käyttäen. Tieto esitetään siten, että puun lehdet ovat joko tiedostoja tai palasia tiedostosta ja niiden vanhemmat joko hakemistoja tai listoja tiedoston muodostavista

osista. Tiedostot pilkotaan tyypillisesti alle 256 kilotavun kokoisiksi palasiksi, jolloin sekä vähennetään tiedon kahdentamista kun sitä muutetaan että helpotetaan palojen jakamista vertaisten välillä. Tämä rakenne mahdollistaa myös puiden järjestämisen versioiksi siten, että versio viittaa sekä yhteen versioon hakemistosta että edelliseen versioon. [Benet 2014]

Kuvassa 3 on puu, joka esittää kahta versiota samasta hakemistosta, sillä erolla että toisessa versiossa yhtä tiedostoista on muokattu. Koska puun solmuihin viitataan niiden tiivisteillä, voidaan automaattisesti käyttää jo olemassa olevia versioita solmuista, jotka eivät ole muuttuneet. [Benet 2014]

Vertaisten löytämiseen IPFS on valinnut hajautetun tiivistetaulun. Tarkalleen ottaen kyse on *hajautetusta huolimattomasta tiivistetaulusta* (DSHT, distributed sloppy hash table). Tällaiset tiivistetaulut muodostuvat useista samankeskisistä DHT-verkoista, jolloin kuorma jakaantuu tasaisemmin. [Benet 2014; Freedman *et al.* 2004]



Kuva 3. Kaksi versiota samasta hakemistosta. Versiossa 2 yhden tiedoston yhtä osaa on muokattu. Hakemistot on esitetty IPFS:n merklepuussa. [Benet 2014]

Kun käyttäjä hakee tiedostoa tietyllä tiivisteellä, etsitään ensin tiivistettä avaimena käyttäen DHT:stä sopivat vertaiset. Tämän jälkeen sovitaan näiden vertaisten kanssa tiedon siirtämisestä. IPFS käyttää siirron neuvotteluun BitSwap -nimistä protokollaa. BitSwap toimii ikään kuin kauppapaikkana, jossa vertaiset vaihtavat tiedostojen palasia keskenään. Protokolla on teknisesti hyvin samankaltainen kuin BitTorrent, paitsi että kaikki vertaiset toimivat samassa parvessa, eivätkä jokainen omassa tiedostokohtaisessa parvessaan. Tyypillisesti vertaiset pitävät kirjaa jaetun

ja ladatun tiedon määrästä, suosien luotettuja ja dataa vastaavasti jakavia lataajia. [Benet 2014]

5. Dat

Dat [Robinson *et al.* 2018; Ogden *et al.* 2017] on erityisesti datatieteessä käytetty, versioitu tiedostonjakoprotokolla. Se on suunniteltu erityisesti tukemaan suuria ja nopeasti päivittyviä datamääriä. Dat pyrkii mahdollistamaan hyvin suurten tietojoukkojen synkronoimisen tehokkaasti, välttäen tarvetta jakaa kaikkea metatietoa etukäteen. Protokolla myös mahdollistaa tehokkaan hakemisen vain osasta dataa ja muutosten reaaliaikaisen seuraamisen.

5.1. Käyttäminen

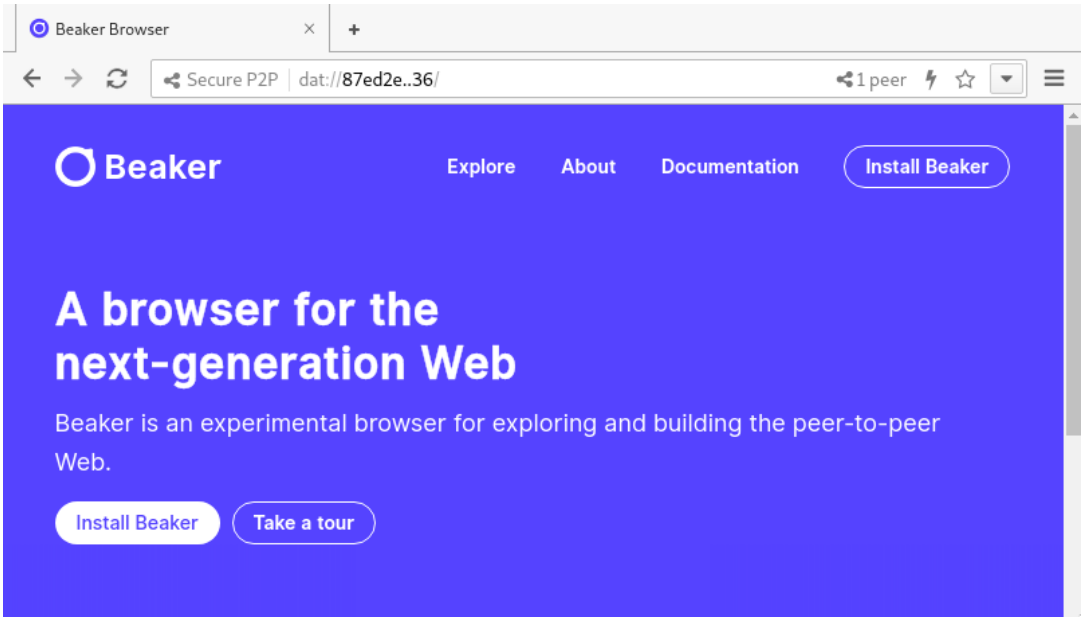
Dat käyttää tiedon perusyksikkönä arkistoa, joka on salattu omistajan yksityisellä avaimella. Arkistoihin viitataan osoitteella, jonka muoto on *dat://(arkiston julkinen avain)*. Vaikka linkit siis näyttävät samankaltaisilta kuin IPFS:n käyttämät linkit, ne viittaavat arkistoihin joiden sisältö kykenee muuttumaan, eivätkä tiettyyn tiedostoversioon. Koska vertaisia etsittäessä arkiston julkinen avain salataan, vain käyttäjät, joilla on kyseinen julkinen avain, ovat tietoisia kyseisen arkiston olemassaolosta. [Ogden *et al.* 2017]

Arkistoa luodessa Dat-sovellus generoi hakemistoon piilotetun .dat-nimisen alikansion, johon tallennetaan metatietoa arkistosta. Käyttäjän kotikansioon taas generoidaan arkistolle yksityinen avain. Tällä yksityisellä avaimella allekirjoitetaan jaettava tieto ja siihen myöhemmin tulevat muutokset. Tämän jälkeen käyttäjät, joilla on julkinen avain hallussaan, voivat sekä päästä käsiksi tiedostoihin että varmentaa tiedostojen oikeellisuuden. [Ogden *et al.* 2017]

Arkistoa ladattaessa käyttäjä tarvitsee ulkopuolista turvallista kanavaa käyttäen arkiston julkisen avaimen eli arkiston osoitteen. Julkista avainta käyttäen hän voi kahdentaa arkiston tai osia siitä muilta vertaisilta, jotka tuntevan saman avaimen. Arkistoon tulevia muutoksia on myös mahdollista seurata reaaliaikaisesti. [Ogden *et al.* 2017]

5.2. Beaker Browser

Vaikka Dat on ensisijaisesti kehitetty tieteellisten datajoukkojen käsittelyyn, sen ympärille on kehittynyt muunkinlaisia sovelluksia. Yksi tämän tutkielman kannalta kiinnostava sovellus on *Beaker Browser* [Beaker Browser 2018]. Beaker on verkkoselain, joka tukee sekä Dat-arkistojen selaamista että mahdollistaa omien arkistojen luomisen ja julkaisemisen suoraan sen käyttöliittymästä. Tämä mahdollistaa Datin käyttämisen IPFS:n tavoin muodostamaan hajautetun vaihtoehdon internetille. Kuvassa 4 on Beaker Browserin etusivu dat-arkistona avattu selaimessa itsessään.



Kuva 4. Kuvankaappaus Beaker Browser -selaimesta. [Beaker Browser 2018]

Beaker Browserin ympärille on myös kehittymässä ekosysteemi hajautettu verkkosovelluksia. Hiljattain julkaistu Fritter on prototyyppi sosiaalisen median sovelluksesta, jossa käyttäjän profiili, viestit ja muut toiminnot tallennetaan käyttäjän Dat-arkistoon. Näin profiilit elävät Datin vertaisverkossa, kuitenkin siten että alkuperäinen käyttäjä voi arkistoa muokaten lähettää uusia viestejä seuraajilleen. Fritter-verkkosovellus käyttää Beaker-selaimen tarjoamia rajapintoja arkistojen lukemiseen ja muokkaamiseen. [Fritter 2018].

5.3. Tekninen kuvaus

Dat-protokolla itsessään on riippumaton tiedonsiirtoprotokollasta tai vertaisten löytämiseen käytetystä mekanismista. Viitetoteutus käyttää oletuksena siirtoon HTTP-protokollaa tai tarvittaessa TCP tai UTP -protokollia. Vertaisten löytämiseen käytetään ensisijaisesti DNS nimipalvelimia tai tarvittaessa Kademlia DHT [Maymounkov *et al.* 2002] -protokollaa. Koska DHT:n käyttäminen voi kuitenkin paljastaa informaatiota käyttäjästä, tämän käyttö ei ole välttämätöntä. [Ogden *et al.* 2017]

Itse tiedon synkronointiin yhteyksien löydyttyä käytetään erillistä *Hypercore*-nimistä moduulia, jota on myös mahdollista käyttää erikseen minkä tahansa datavirran synkronoimiseen. Hypercoren päällä on erillinen *Hyperdrive*-protokolla, joka toteuttaa Datin arkistojen synkronoinnin. [Ogden *et al.* 2017]

Dat säilyttää tiedon arkistoissa, joiden muutoshistoria tallennetaan. Oletuksena käyttäjät kuitenkin lataavat ja säilyttävät vain uusimman version levytilan säästämiseksi. Historiatietoa säilyttävät tyypillisesti vain suurempitehoiset palvelinlaitteet, jotka voivat tarjota edellisiä versioita pyydettäessä. [Ogden *et al.* 2017]

Tiedon kuljettamiseen ja tallentamiseen Dat käyttää omaa tiedostomuotoaan nimeltä SLEEP. Samoin kuin IPFS, myös Dat järjestää tiedostot merklepuuhun. Toisin kuin IPFS, Dat järjestää tiedostot itsenäisiin salattuihin arkistoihin, tiedostojen

palaset eivät muodosta samanlaista globaalia nimiavaruutta kuin IPFS. Näin ollen ainoastaan vain arkiston tuntevat vertaiset voivat tarjota palasta, vaikka se olisikin osa muita arkistoja. [Ogden *et al.* 2017]

Vaikka tiedostojen data paloitellaan merklepuuhun lähes samoin kuin IPFS toimii, Dat ei koodaa tiedostopolkuja samaan puuhun, jossa data on. Sen sijaan erillisessä merklepuussa on esitetty metadatarekisteri, johon kirjataan tiedoston polut ja nimet, sekä niiden sijainti varsinaisessa merklepuussa. Tämä ratkaisu sallii metadatan hakemisen erillään itse tiedostoista mahdollistaen osittaiset lataukset keskeltä puuta. Lataaja voi siis ensin etsiä metadataa lukien haluamiensa tiedostojen sijainnin arkistossa, ja pyytää vain tarvitsemansa solmut merklepuusta. Datan siirtoon Datilla on taustalla olevasta alustasta riippumaton tilaton viestiprotokolla, jonka viestit on koodattu protobuf-formaatissa. [Ogden *et al.* 2017]

6. Vertailua

BitTorrent, IPFS ja Dat mahdollistavat kaikki tiedostojen jaon vertaisten välillä. Näillä kaikilla on myös teknisellä tasolla paljon yhteistä: kaikki mahdollistavat vertaisten löytämisen DHT:n avulla, ja ovat siten luokiteltavissa toisen sukupolven jäsennellyiksi vertaisverkoiksi. Kaikki myös jossain kohtaa protokollaa tunnistavat tiedostot niiden tiivisteiden avulla, joskin IPFS vie ajatuksen pidemmälle kuin muut.

Erilaiset käyttötarkoitukset kuitenkin johtavat erilaisiin valintoihin, kun verkkoja vertailee tarkemmin. Taulukossa 1 on kuvattu tiivistetysti joitakin keskeisiä eroavaisuuksia. Seuraavissa kohdissa vertailen kaikkia kolmea paria tarkemmin.

Ominaisuus	BitTorrent	IPFS	Dat
Vertaisten löytäminen	Seurantapalvelin	DSHT	DNS
Tyypillisin käyttökohde	Piratismi	P2P internet	Suuret datajoukot
Muuttuvaa dataa	Ei	Ei	Kyllä

Taulukko 1. Vertaisverkkojen keskeisten ominaisuuksien vertailu

6.1. BitTorrent ja IPFS

BitTorrent keskittyy jakamaan yksittäisiä suuria tietojoukkoja, ja IPFS puolestaan pyrkii luomaan globaalisti hajautetun tiedostojärjestelmän. Teknisesti niillä on kuitenkin paljon yhteistä. Jos BitTorrentia käyttää DHT:n kautta, molemmat protokollat sekä löytävät vertaiset että vaihtavat palasia hyvin samankaltaisesti. Keskeinen ero kuitenkin on, että koko IPFS-ekosysteemi toimii yhden suuren parven sisällä, kun BitTorrentissa jokainen parvi on itsenäinen. Tästä seuraa esimerkiksi, etteivät parvet kykene tekemään yhteistyötä, mikäli sama tiedosto tunnetaan useassa parves-

sa. Koska IPFS:ssä on vain yksi parvi, identtisen tiedoston voi ladata minkä tahansa merklepuun osana.

Protokollien päälle rakennetut järjestelmät ovat luonteeltaan erilaisia. IPFS pyrkii olemaan alusta sovelluksille, kun taas BitTorrentilla on oma keskeinen sovelluksensa johon se on optimoitu.

6.2. BitTorrent ja Dat

Sekä BitTorrent että Dat jakavat tiedostot arkistoihin, joiden sisällä tiedon jako tapahtuu. Dat kuitenkin mahdollistaa myös tiedon muuttumisen arkistojen sisällä, kun taas BitTorrent vaatii uuden parven muodostamista uutta versiota varten. BitTorrent on myös tarkoitettu koko tietojoukon lataamiseen kerralla, kun taas Dat pyrkii mahdollistamaan vain tiettyjen osien lataamisen. BitTorrent on näistä kahdesta sekä vanhempi että käytetympi, sekä siten myös käytännössä luotettavampi.

Samoin kuin Datia, myös BitTorrentia on käytetty tieteellisten datajoukkojen siirtämiseen [Cohen *et al.* 2014]. BitTorrent kuitenkin soveltuu vain staattisen datajoukkon kopioimiseen, eikä esimerkiksi muuttuvan datan seuraamiseen kuten Dat.

6.3. IPFS ja Dat

IPFS ja Dat ovat monilta osin hyvin samantyyppisiä. Molemmat ovat melko uusia, merklepuiden päälle rakennettuja järjestelmiä. Toisaalta niiden käyttötavat poikkeavat toisistaan, sillä IPFS pyrkii rakentamaan globaalin linkittyneen verkon, kun taas Dat on kiinnostunut yksittäisten arkistojen tehokkaasta jakamisesta internetin yli. Nämä poikkeavat tavoitteet johtavat erilaisiin teknologisiin valintoihin toteutuksen yksityiskohdissa.

Sekä Dat että IPFS toteuttavat jonkinlaisen versioinnin tiedostoilleen, joskin hyvin eri tasoilla. IPFS:ssä versiot rakennetaan taustalla olevan merklepuun päälle sovellustasolla, kun Datissa ne taas ovat kiinteä osa protokollaa. Vaikka IPFS:llä on erilaisia tekniikoita muuttuvan tiedon esittämiseen, se ei ole samalla tavalla osa protokollaa kuin Datin muuttuvat arkistot.

Sovelluksilla on myös erilainen lähestymistapa yksityisyyteen. Ellei sisältöä itse salaa etukäteen, IPFS:n merklepuuhun julkaistu data on julkista, sillä palasten tiivisteet julkaistaan DHT:ssa vertaisille [Benet 2014]. Dat sen sijaan salaa lähtökohtaisesti liikenteen, ja tiedostosta ja sen lataajista ovat tietoisia vain muut, joilla on arkiston julkinen avain eli sen osoite [Ogden *et al.* 2017].

IPFS on myös vahvemmin sidoksissa DSHT-toteutukseensa, kun Dat taas ei ota protokollatasolla kantaa yhteyksien löytämiseen. Tästä seuraa että IPFS on yhtenäisempi verkko, kun taas Dat ei voi taata, että erilaiset vertaisten hakumekanismit kykenevät toimimaan yhdessä. Toisaalta Dat pystynee sopeutumaan paremmin teknologiassa tapahtuvaan kehitykseen, sillä se voi helpommin omaksua uusia mekanismeja vertaisten yhdistämiseen.

IPFS:n ydinkäyttötapauksen eli hajautetumman internetin toteuttamisen mahdollistaa jokseenkin samalla ajatuksella Beaker Browser. Molemmat mahdollistavat selaimella navigoimisen verkko-osoitteeseen siten, että sivun sisältö haetaan parvelta. Linkkien merkitys kuitenkin vaikuttaa muodostuvaan ekosysteemiin: Dat tukee paremmin dynaamista sisältöä kuten moderneja websovelluksia, kun taas IPFS muodostaa yksinkertaisemman, dokumenttipohjaisen verkon.

7. Yhteenveto

Tässä tutkielmassa tutustuin kolmeen vertaisverkkoteknologiaan, joilla voi jakaa tiedostoja vertaisverkon kautta vertaiselta toiselle. Kävin läpi niiden taustaa, käyttämistä ja teknistä toteutusta sekä vertailin niitä keskenään.

BitTorrent soveltuu hyvin suurten, muuttumattomien datajoukkojen jakamiseen. IPFS ja Dat ovat vielä uudempia järjestelmiä, eivätkä siten välttämättä yhtä luotettavia. Niillä lienee kuitenkin mielenkiintoisia käyttökohteita tulevaisuudessa. Näitä käyttökohteita saattavat olla esimerkiksi luotettavampi arkistointi sekä uudenlaisten verkkosovelluksien kehittäminen. IPFS:llä ja Datilla on myös mekanismeja toistensa käyttökohteiden omaksumiseen, joten on mahdollista että vain toinen niistä leviää laajempaan käyttöön.

Vertailu on osin puutteellinen. Suuri käytännön merkitys on järjestelmien suorituskyvyllä, mutta niiden testaaminen ei kuulunut tämän tutkielman laajuuteen. Uusina järjestelminä näiden vertaisverkkojen välillä ei ole tehty tieteellisiä suorituskykyvertailuja, joista olisin tietoinen. Myöskin sovellusten käytettävyyttä olisi syytä vertailla tarkemmin, sillä se vaikuttaa uusien teknologioiden käyttöönottoon.

Tämä tutkielma ei ole myöskään kattava katsaus tiedostojen jakoon vertaisverkoissa, sillä erilaisia protokollia on lukuisia. Näiden kolmen valitun kautta on kuitenkin mahdollista saada jonkinlainen näkemys teknologian kehityksestä ja tulevasta suunnasta.

Viitteet

Sawood Alam, Mat Kelly and Michael L. Nelson. 2016. InterPlanetary Wayback: The permanent web archive. In: *Proc. of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries, JCDL '16*, 273–274.

Stephanos Androutsellis-Theotokis and Diomidis Spinellis. 2004. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.* 36, 4.

Beaker Browser. 2018. Browsing with Beaker. `https://beakerbrowser.com/docs/tour/#browsing-with-beaker`. Checked 15.11.2018.

Beaker Browser. 2018. Fritter – A peer-to-peer social feed app. `https://github.com/beakerbrowser/fritter/`. Checked 15.11.2018.

Juan Benet. 2014. IPFS – content addressed, versioned, P2P file system. `https://arxiv.org/pdf/1407.3561`. Checked 01.10.2018.

Bram Cohen. 2003. Incentives build robustness in BitTorrent. In: *Proc. of the 1st Workshop on Economics of Peer-to-Peer systems*, 6, 68–72.

Joseph Paul Cohen and Henry Z. Lo. 2014. Academic torrents: A community-maintained distributed repository. In: *Proc. of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, Article 2, 2 pages.

Michael J. Freedman, Eric Freudenthal, and David Mazières. 2004. Democratizing content publication with Coral. In: *Proc. of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 239–252.

Greg Hazel and Arvid Norberg. 2018. BitTorrent enhancement proposal 9: Extension for peers to send metadata files. `http://www.bittorrent.org/beps/bep_0009.html`. Checked 15.11.2018.

Jari Kattelus. 2008. *P2P vertaisverkot – Uhka vai mahdollisuus?.* Pro gradu - tutkielma. Tietojenkäsittelytieteiden laitos. Tampereen yliopisto.

Andrew Loewenstern and Arvid Norberg. 2008. BitTorrent enhancement proposal 5: DHT protocol. `http://www.bittorrent.org/beps/bep_0005.html`. Checked 15.11.2018.

Petar Maymounkov, and David Mazières. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In: *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, 53—65.

Maxwell Ogden, Karissa McKelvey, Mathias Buus Madsen and Code for Science. 2017. Dat – Distributed dataset synchronization and versioning. `https://osf.io/fp2sy`. Checked 01.10.2018.

Danielle C. Robinson, Joe A. Hand, Mathias Buus Madsen, and Karissa R. McKelvey. 2018. The Dat Project, an open and decentralized research data tool. *Sci Data* 5, 180221.

Transmission Project. 2005. Transmission – A fast, easy and free BitTorrent client. `https://transmissionbt.com/`. Checked 12.12.2018.

Matteo Varvello, Moritz Steiner and Koen Laevens. 2012. Understanding BitTorrent: A reality check from the ISP's perspective. *Computer Networks* 56, 3, 1054–1065.

Paul A. Watters, Robert Layton and Richard Dazeley. 2011. How much material on BitTorrent is infringing content? A case study. *Information Security Technical Report* 16, 2, 79–87.

Katsaus käänteismatriisin määrittämisessä käytettäviin perusalgoritmeihin

Chi-Hao Lay

Tiivistelmä.
Tutkielmassa käsittelemme käänteismatriisien laskemisessa hyvin tunnettuja perusalgoritmeja Gaussin-Jordanin eliminointimenetelmää, LU-hajotelmaa sekä Newtonin menetelmää. Uusien löytöjen sijasta näytämme esimerkein, kuinka kyseiset algoritmit toimivat ja katsomme, mitä niistä on kirjoitettu.

Avainsanat ja -sanonnat: Gaussin-Jordanin eliminointimenetelmä, käänteismatriisi, LU-hajotelma, Newtonin menetelmä

1. Johdanto

Käänteismatriiseista tietoa etsiessä saattaa törmätä suositukseen, että käänteismatriisia ei yleensä tarvita. Du Croz ja Higham [1992] lainaavatkin Forsythen, Malcomin ja Molerin [1977] toteamusta kyseisestä suosituksesta. Myös Haataja ja muut [1999] suosittelevat korvaamaan käänteismatriisin käytön muilla menetelmillä. Tulee kuitenkin huomata, että suositus on vanha. Aihe vaikuttaakin mielestämme usein käsiteltävän lyhyesti lineaaristen yhtälöryhmien sivuseikkana.

Käänteismatriisialgoritmeja tunnetaan useita, mutta perehdymme vain muutamaan niistä. Tämä voi olla oikein riittävä, sillä Householderin [1964, 122] näkemyksen mukaan monet algoritmit vaikuttaisivat perustuvan muutamaan perusalgoritmiin. Käsittelemämme algoritmit ovat siis jo pitkään tunnettuja ja niistä on myös laajalti kirjoitettu. Käänteismatriisin määritystä monipuolisemmin ja täsmällisemmin käsittelevät muun muassa Turing [1948], Fox [1950], Greenspan [1955] ja Wilkinson [1961].

Iteratiivisista menetelmistä on usein mainittu *Gaussin-Seidelin menetelmä* [Wilkinson 1961; Householder 1964, 112]. Tutkielmassa käsiteltyjen menetelmien lisäksi Householder [1964] käsittelee menetelmiä *Shermanin-Morrisonin kaava*, *Choleskyn menetelmä*, *Croutin menetelmä* ja *matriisien ositus* (method of partitioning). Greenspan [1955] käsittelee joidenkin edellä mainittujen lisäksi *Framen menetelmän* ja joitakin nimeämättömiä. Hän mainitsee myös *Monte Carlon menetelmän*. Wikipediaan [2018] on koottu myös *Cayleyn-Hamiltonin lause*, *ominaishajotelma* (eigendecomposition) ja *von Neumannin sarja*. *Strassenin algoritmilla* saa myös laskettua käänteismatriisin [Bailey and Ferguson 1988].

Tutkielman keskiössä ovat esimerkit, jotka itse ovat tarkastelun kohteena. Esimerkkien matriisit on laskettu GNU Octavea apuna käyttäen. Koska laskimme tietokoneohjelmistolla, lukijan on hyvä olla tietoinen, että esimerkit voivat poikesta tarkasta arvosta.

Luvussa 2 aloitamme esitietojen kertaamisella. Sen jälkeen siirrymme algoritmeihin, jotka perustuvat Gaussin eliminointimenetelmään. Näitä ovat luvussa 3 esitetty Gaussin-Jordanin algoritmi ja luvussa 4 LU-hajotelmat. Luvussa 5 käännämme matriisin Newtonin menetelmällä. Luku 6 on yhteenveto.

2. Esitietoja

Määrittelemme ensiksi *matriisin* käsitteen. Merkitsemme isolla kirjaimella A matriisia, jonka koko on $m \times n$. Ajattelemme tutkielman ajan sitä kaksiulotteisena taulukkona. Käytämme alkiolle a_{ij} paikassa (i, j) merkintää $A(i, j)$ eli matriisi

$$A = \begin{bmatrix} A(1,1) & \dots & A(1,n) \\ \vdots & & \\ A(m,1) & & A(m,n) \end{bmatrix}.$$

Lyhyemmin voisimme merkitä $A \in \mathbb{R}^{m \times n}$. [Golub and van Loan 1996, 3.] Perehdymme algoritmeihin reaalilukumatriisien avulla. Tutkielmassa käsittelemme vain matriiseja, jotka ovat kokoa $n \times n$. Näitä matriiseja kutsumme *neliömatriiseiksi* [Pitkäranta 2015, 663].

2.1. Merkinnöistä

Osassa lähdekirjallisuudessa on pseudokoodin yhden rivin `for`-silmukan operaatioiden ilmaisuissa tiivistetty esimerkiksi muotoon $A(i : n, j)$, mikä viittaa sarakeen j alkioihin, jotka sijaisevat rivistä i riviin n. Tätä tyyliä käyttävät muun muassa Golub ja van Loan [1996]. Voimme esimerkiksi kasvattaa sarakkeen j alkioiden arvoa yhdellä merkitsemällä

$$A(1 : n, j) + 1 = [A(1, j) + 1, \quad A(2, j) + 1, \quad \dots, \quad A(n, j) + 1].$$

Joillekin matriiseille on olemassa tiiviimpiä tallennustapoja, jolloin algoritmeja täytyy muokata lukemaan indeksit oikein. Esimerkiksi symmetrisen matriisin voimme esittää vektorina [Golub and van Loan 1996, 21]. Manipuloitua matriisia merkitsemme yläindeksinä $A^{(k)}$, jota käytetään paljon. Merkintätavassa luku k kertoo, montako kertaa matriisia on muutettu. Matriisien A ja B yhdistettyä matriisia merkitsemme lohkomatriisina $[A\ B]$.

2.2. Nimettyjä matriiseja

Seuraavaksi kertaamme joitakin lineaarialgebran käsitteitä. Pitkäranta [2015, 664] määrittelee matriisin *transpoosin* $A^\top$, jossa alkiolle pätee

$$A^\top(i, j) = A(j, i).$$

Voimme siis ajatella, että alkiot $A(i,j)$ asetetaan paikkaan $A^\top(j,i)$, toisin sanoen alkiot vaihtavat paikkaa keskenään määritelmän mukaisesti. Edelleen, hän määrittelee *symmetrisen matriisin* ehdoksi

$$A^\top = A,$$

mikä tarkoittaa samaa kuin, että matriisin kaikille alkioille pätee $A(i,j) = A(j,i)$.

Yläkolmiomatriisin U alkioilla $U(i,j) = 0$, kun $i > j$, ja vastaavasti *alakolmiotriisilla* L, kun $i < j$. Matriisia D, jossa alakolmio- ja yläkolmiomatriisien ehdot ovat voimassa samaan aikaan kutsumme *diagonaalimatriisiksi*. Siis $D(i,j) = 0$, kun $i \neq j$. [Pitkäranta 2015, 676–677.]

Määrittelemme *käänteismatriisin* A^{-1} yhtälöllä

$$AX = I,$$

missä A on kääntyvä neliömatriisi kokoa $n \times n$, ja I samankokoinen identiteettimatriisi eli diagonaalimatriisi, jonka alkiot ovat 1. Jos on olemassa sellainen matriisi $X = A^{-1}$, että yhtälö pätee, niin kutsumme sitä käänteismatriisiksi. [Golub and van Loan 1996, 50; Pitkäranta 2015, 670.] Kyseinen määritelmä ei siis päde kaikille matriiseille. On kuitenkin määritelty yleistettyjä käänteismatriiseja, jotka pätevät kaikille matriiseille. Näitä emme kuitenkaan käsittele tässä tutkielmassa.

2.3. Gaussin eliminointimenetelmä

Monissa oppimateriaaleissa opetetaan yleensä *Gaussin eliminointimenetelmää* lineaaristen yhtälöryhmien kanssa, joiden yhteydessä saattaa olla esitelty myös joitakin matriisihajotelmia, kuten luvun 4 LU-hajotelma. Esimerkiksi Pitkäranta [2015] esittää aiheen tällä tavalla ja myös Haataja ja muut [1999]. Tarkastelemme Gaussin eliminointimenetelmää lähinnä osana muita algoritmeja. Muun muassa Wolfram [2018] -ohjelmointikielen funktioiden `LUDecomposition(M)` ja `Inverse(M)` toteutukset hyödyntävät algoritmia osittaistuennalla.

Algoritmi 1 Gaussin algoritmi ilman tuentaa. [Haataja *ym.* 1999, 34.]

Oletukset: Matriisi $A \in R^{n \times n}$ ja rivien määrä n.
Tulos: Manipuloitu matriisi A, joka sisältää vastaavan yläkolmiomatriisin U.

1: **procedure** GAUSS(A, n)
2: **for** $k = 1 : n - 1$ **do** ▷ Valitsemme ensin sarakkeen k.
3: **for** $i = k + 1 : n$ **do** ▷ Valitsemme, sitten rivit i.
4: $r \leftarrow A(i, k)/A(k, k)$
5: $A(i, k + 1 : n) \leftarrow A(i, k + 1 : n) - r \cdot A(k, k + 1 : n)$
6: **end for**
7: **end for**
8: **end procedure**

Gaussin eliminointimenetelmässä manipuloimme rivioperaatioilla matriisia A vastaavaksi yläkolmiomatriisiksi [Haataja *ym.* 1999, 34]. Algoritmista 1 on jätetty pois lähdeteoksen takaisinsijoitusvaihe ja operaatioiden suoritus vektorille $\mathbf{b}$, joka tulisi riville 6 silmukan sisälle. Jatkossa, kun mainitsemme Gaussin algoritmin yhdistetyn matriisin yhteydessä, tarkoitamme silloin, että se sisältää operaation suorituksen matriisille B. Algoritmista 1 voisimme ilmeisesti tehdä toisen version korvaamalla rivin 5 lauseekkeella

$$[A\ B](i, k + 1 : 2n) \leftarrow [A\ B](i, k + 1 : 2n) - r \cdot [A\ B](k, k + 1 : 2n),$$

missä $[A\ B] \in \mathbb{R}^{n \times 2n}$. Tämä seuraa siitä, että matriisilla on nyt $2n$ saraketta, ja rivioperaatio suoritetaan nyt myös matriisille B. Algoritmissa 2 viittaamme Gaussin algoritmilla tähän versioon.

Esimerkki 2.1. Havainnollistamme nyt Gaussin algoritmia. Huomaamme, että algoritmi 1 ei vaikuta oikeasti nollaavan alapuolisia alkioita. Tarkistamme siis huomion. Olkoon

$$A = \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 4.5 & 4.7 \\ 5.4 & 8.8 & 5.1 \end{bmatrix}.$$

Kun $k = 1$ ja $i = k + 1 = 2$, niin rivi 5 on seuraavanlainen

$$A(2, 2 : 3) = A(2, 2 : 3) - r \cdot A(1, 2 : 3).$$

Alkio $A(2, 1)$ näyttäisi tosiaan jäävän laskematta. Ensimmäisen silmukan jälkeen saamme matriisin

$$A^{(1)} = \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 4.5 - \frac{1.0}{6.1} \cdot 9.6 & 4.7 - \frac{1.0}{6.1} \cdot 2.1 \\ 5.4 & 8.8 & 5.1 \end{bmatrix} \approx \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 2.9 & 4.4 \\ 5.4 & 8.8 & 5.1 \end{bmatrix}.$$

Joudumme pyöristämään alkiot $A(2, 2) \approx 2.926$ ja $A(2, 3) \approx 4.356$, jotta ne mahtuisivat mukavasti sivulle. Olkoon tämä myös esimerkkinä pyöristyksen laskentavir-

heestä ja sen käytöstä myöhemmässä vaiheessa. Nyt $k = 1$ ja $i = 3$

$$A^{(2)} = \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 2.9 & 4.4 \\ 5.4 & 8.8 - \frac{5.4}{6.1} \cdot 9.6 & 5.1 - \frac{5.4}{6.1} \cdot 2.1 \end{bmatrix} \approx \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 2.9 & 4.4 \\ 5.4 & 0.3 & 3.2 \end{bmatrix}.$$

Sitten $k = 2$ ja $i = 3$

$$A^{(3)} = \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 2.9 & 4.4 \\ 5.4 & 0.3 & 3.2 - \frac{0.3}{2.9} \cdot 4.4 \end{bmatrix} \approx \begin{bmatrix} 6.1 & 9.6 & 2.1 \\ 1.0 & 2.9 & 4.4 \\ 5.4 & 0.3 & 2.7 \end{bmatrix}.$$

Vaikka nollaisimme alkiot, niin emme luultavasti aina tarvitse sitä ominaisuutta. Tavallisessa yhtälönryhmän ratkaisussa algoritmi 1 sisältäisi vielä takaisinsijoituksen, jossa tarvitaan ainoastaan yläkolmion alkiot. Tuennan käyttö saattaa kuitenkin vaikuttaa nolla-alkioiden tarpeeseen, koska silloin tapahtuu alkioiden vertailua.

2.4. Tuenta

Tuennassa (pivoting) vaihdamme matriisin sarakkeiden ja rivien paikkoja, jotta saamme halutun tukialkion johonkin paikkaan (i, j). *Osittaistuennassa* (partial pivoting) etsimme tukialkiota samasta sarakkeesta rivejä vaihtamalla. Jos etsimme tukialkiota koko matriisista, niin kutsumme prosessia *täydelliseksi tuennaksi* (complete pivoting). Tuenta muun muassa estää nollalla jakamisen. Näyttää myös siltä, että usein määritelmä sisältää sen, että etsitään suurinta alkiota. [Haataja *ym.* 1999, 35; Pitkäranta 2015, 683.] Tutkielmassa käsitteellä *osittaistuenta sarakkeittain* (by columns) tarkoitamme sarakkeiden vaihtamista tukialkion löytämiseksi [Cortés and Peña 2007].

Gaussin eliminointimenetelmässä (kohta 2.3) voi olla tarpeen hyödyntää tuentamenetelmiä. Tämä siis vastaa rivioperaation rivinvaihtoa, jonka teemme, kun diagonaalialkio $A(i, i)$ on nolla. Cortés ja Peña [2007] vertailevat tuentamenetelmiä. Heidän tarkastelemia menetelmiä ovat osittaistuenta, osittaistuenta sarakkeittain, täydellinen tuenta, *"rook pivoting"*, *kaksinkertainen-* (double), *kolminkertainen-* (triple), ja *nelinkertainen osittaistuenta* (quadruple). Pan ja Zhao [2017] kertovat tuennan hidastavan merkittävästi algoritmin suoritusta ja he tutkivat matriisin esikäsittelyä, jotta esimerkiksi Gaussin algoritmin voisi suorittaa ilman tuentaa. Sopivalla esikäsittelyllä voi siis välttää tuennan käyttämisen.

2.5. Laskentatarkkuudesta

Numeerisessa laskennassa Haatajan ja muiden [1999, 27–28] mukaan laskentavirheitä voivat aiheuttaa muun muassa pyöristys-, katkaisuvirheistä tai siitä, että itse menetelmä ei ole stabiili. *Pyöristysvirhe* (round-off error) aiheutuu, kun laskuoperaation tulos joudutaan pyöristämään. *Katkaisuvirheessä* (truncation error) ei pyöristetä, vaan otetaan osa luvun esitysmuodosta. [Haataja *ym.* 1999, 27–28.] Ajatelemme,

että menetelmä on stabiili, jos se kestää pyöristysvirheet. Ne eivät siis vaikuta merkittävästi lopputulokseen [Butterfield and Ngondi 2016]. Esimerkiksi erittäin suurilla tai pienillä luvuilla voi esiintyä laskentavirheitä johtuen tietokoneen rajallisesta kyvystä käsitellä liukulukuja [Cormen *et al.* 2009, 813]. Cormen ja muut [2009, 813] luonnehtivatkin stabiiliutta tärkeäksi aihealueeksi.

3. Gaussin-Jordanin eliminointimenetelmä

Valitsemme ensimmäiseksi algoritmiksi *Gaussin-Jordanin eliminointimenetelmän*, sillä se on myös usein opetettu, joten koemme, että sitä on nytkin tarpeen esittää. Etenkin, koska käänteismatriisin määrityksestä Turing [1948] luonnehtii Gaussin-Jordanin menetelmää ”todennäköisesti suoraviivaisimmaksi”. Koska tutkielmaa tehdessä ei ole löytynyt yksinkertaisempaa, oletamme väitteen yhä pitävän. Tämä on myös niin sanottu *suora menetelmä* eli algoritmi vaatii rajallisen määrän laskuoperaatioita [Householder 1964, 122]. Siis suoritus päätetään, kun tietty määrä n laskuoperaatioita on suoritettu ja matriisi on silloin käännetty.

Aloitamme prosessin sillä, että muodostamme matriisista A ja identiteettimatriisista I yhdistetyn matriisin $[A\ I]$, jota manipuloimme tilaan, jossa matriisin A tilalle saamme identiteettimatriisin I eli

$$[A\ I] \to [I\ A^{-1}]. \tag{1}$$

Tällöin oikealle muodostuva matriisi A^{-1} on haluamamme käänteismatriisi. Tätä menetelmää kutsutaan myös Gaussin eliminointimenetelmäksi. [Pitkäranta 2015, 689-690.] Tämä vaikuttaisi olevan yleisin tapa esittää menetelmä.

Toinen tapa olisi Turingin [1948] täsmällisempi kuvailu matriisitulon avulla eli

$$A^{(n)} = J_n \cdot J_{n-1} \cdots J_1 \cdot A,$$

missä J_i on jokin matriisi, jolla manipuloimme matriisia eli tavoitelemme $A^{(n)} = I$. Voimme ehkä havainnollistaa sitä laskemalla, jolloin n:n vaiheen jälkeen saamme

$$(J_n \cdot J_{n-1} \cdots J_1) \cdot AX = (J_n \cdot J_{n-1} \cdots J_1) \cdot I.$$

Jos $J_n \cdots J_1 = A^{-1}$, niin

$$(A^{-1}A)X = A^{-1}I \quad \Leftrightarrow \quad IX = A^{-1}$$

eli $X = A^{-1}$. Siis jokaisella J_{i+1} kertomisen jälkeen $J_{i+1} \cdots J_1 A$ lähestyy identiteettimatriisia I kaikilla $i \leq n$. Sivuutamme nyt matriisien J_n määrittämisen ja siirrymme menetelmän hyödyntämiseen. Turing [1948] kutsuu tätä *Jordanin menetelmäksi.* Vastaavasti hän on kuvaillut myös Gaussin algoritmin.

Gaussin-Jordanin algoritmista on kirjoitettu runsaasti, joten vältämme toistoa ja käsittelemme esimerkissä 3.1 erästä versiota, josta käy ilmi algoritmin toimintaperiaate. Muun muassa Greenspan [1955] on esittänyt hyvän esimerkin perustapauksesta. Gaussin-Jordanin algoritmia sanotaan erikoistapaukseksi vastaavasta

menetelmästä

$$\begin{bmatrix} I & A \\ 0 & I \end{bmatrix} \to \begin{bmatrix} P & I \\ 0 & Q \end{bmatrix},$$

mistä lasketaan $A^{-1} = PQ$ [Greenspan 1955].

Algoritmi 2 Gaussin-Jordanin eliminointimenetelmä perustuen Edwards Jr.:n ja Penneyn [1988, 24] algoritmikuvaukseen.

Oletukset: Matriisi $[A\ I] \in \mathbb{R}^{n \times 2n}$ ja rivien määrä n.
Tulos: Matriisi $[A\ I]$ muodossa $[I\ A^{-1}]$.

1: **procedure** GAUSS-JORDAN($[A\ I], n$)
2: GAUSS($[A\ I], n$) ▷ Matriisi A yläkolmiomatriisiksi.
3: **for** $k = 1 : n$ **do** ▷ Jaamme rivit vastaavalla diagonaalialkiolla.
4: $[A\ I](k, k : 2n) \leftarrow [A\ I](k, k : 2n)/[A\ I](k, k)$
5: **end for**
6: Eliminoi rivioperaatioilla alkiot $[A\ I](i, j)$ nolliksi, kun $i < j \leq n$.
7: **end procedure**

Algoritmista 2 näemme, että kyseessä on käytännössä Gaussin algoritmi ja sen soveltaminen toisinpäin. Voimme myös päätellä, että on mahdollista ohittaa tiettyjen alkioiden laskeminen kuten algoritmissa 1. Jos algoritmi todellakin ilmaisee tätä, niin olisi varmaan mahdollista myös ensin eliminoida alkiot nolliksi ja sitten suorittaa rivilä 3 alkavan `for`-silmukan.

Kuten Gaussin algoritmissa, myös Gaussin-Jordanin algoritmissa voi olla tarpeen hyödyntää tuentaa. Emme voi kuitenkaan käyttää mitä tahansa tuentamenetelmää, sillä esimerkiksi Cortésin ja Peñan [2007] mukaan ”on hyvin tunnettu”, että Gaussin-Jordanin algoritmi on epästabiili osittaistuennalla (riveittäin). He mainitsevat myös, että Dekker ja Hoffman [1989] osoittaa tuennan sarakkeittain stabiiliksi. Cortés ja Peña [2007] päättelevät, että osittaistuenta sarakeittain, kaksinkertainen- ja nelinkertainen tuenta soveltuvat käytettäväksi Gaussin-Jordanin algoritmin suorittamisessa.

Peters ja Wilkinson [1975] selvittävät Gaussin-Jordanin algoritmin stabiiliutta yhtälöryhmien ratkaisemisessa. Heidän mukaan algoritmin voi ajatella kahtena osana: Gaussin eliminointina ja jälkimmäisenä eliminointivaiheena (vertaa algoritmin 2 vaiheet 3–6). Petersin ja Wilkinsonin [1975] mukaan on riittävä analysoida pelkästään jälkimmäinen vaihe. Loppupäätelmä on, että yhtälönratkaisussa Gaussin-Jordanin algoritmi tuottaa usein suurempia arvoja kuin Gaussin eliminointimenetelmä takaisinsijoituksella. Emme ole kuitenkaan varmoja, pätevätkö samat tulokset myös matriisiyhtälöille.

McLean [1980] esittelee muokattua Gaussin-Jordanin algoritmia, jolla on tarkoitus kiertää virhe, missä merkitseviä numeroita katoaa suorituksen aikana. Menetelmässä algoritmia sovelletaan matriisiin $A' = [A\ \mathbf{b}\ I]$, jossa vektorin $\mathbf{b}$ alkiot

koostuvat vastaavan rivin alkioiden summan vastaluvusta

$$\mathbf{b}(i) = -\sum_{j=1}^{n} A(i,j).$$

Merkitsevien numeroiden kadotessa jollekin alkiolle vaiheessa k, voimme laskea arvon uudelleen seuraavasti

$$A^{(k)}(i,j) = -[\mathbf{b}^{(k)}(i) + \sum_{l=1, l\neq j}^{n} A^{(k)}(i,l)].$$

Esimerkki 3.1. Näytämme kuinka menetelmä toimii. Oletetaan, että voimme esittää vain kahden desimaalin tarkkuudella. Käytämme esimerkkinä *alimatriisia* esimerkistä 2.1. Olkoon

$$A = \begin{bmatrix} 6.1 & 9.6 \\ 1.0 & 4.5 \end{bmatrix}.$$

Tällöin laajennettu matriisi on

$$A' = \begin{bmatrix} 6.1 & 9.6 & -15.7 & 1 & 0 \\ 1.0 & 4.5 & -5.5 & 0 & 1 \end{bmatrix}.$$

Suoritamme siis Gaussin eliminoinnin ensimmäiseksi. Tällä kertaa käsittelemme nollatut alkiot, sillä niitä tarvitaan vektorin $\mathbf{b}$ esittämisessä ja kadonneen alkion laskemisessa. Ensimmäisessä vaiheessa

$$A'^{(1)} = \begin{bmatrix} 6.1 & 9.6 & -15.7 & 1 & 0 \\ 0 & 2.9262 & -2.9262 & -0.16393 & 1 \end{bmatrix}.$$

Vaikka emme kadottaneet merkitseviä numeroita, niin selvästi

$$A(2,2) = -(-2.9262 + 0) = 2.9262.$$

Manipuloimme nyt diagonaalialkiot ykkösiksi. Se hoituu jakamalla rivi vastaavalla diagonaalialkiolla

$$A^{(2)} = \begin{bmatrix} 1 & 1.5738 & -2.5738 & 0.16393 & 0 \\ 0 & 1 & -1 & -0.056021 & 0.34174 \end{bmatrix}.$$

Edelleen näemme $\mathbf{b}^{(2)}$ alkioiden pitävän paikkansa. Nyt suoritamme Gaussin eliminoinnin ”toisinpäin”, jolloin

$$A^{(3)} = \begin{bmatrix} 1 & 0 & -2.5738 - (\frac{1.5738}{1} \cdot (-1)) & 0.16393 - \frac{1.5738}{1} \cdot (-0.056021) & 0 - \frac{1.5738}{1} \cdot 0.34174 \\ 0 & 1 & -1 & -0.056021 & 0.34174 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & -1 & 0.25210 & -0.53783 \\ 0 & 1 & -1 & -0.056021 & 0.34174 \end{bmatrix}.$$

Saamme siis käänteismatriisiksi

$$A^{-1} = \begin{bmatrix} 0.25210 & -0.53783 \\ -0.056021 & 0.34174 \end{bmatrix}.$$

4. LU-, LDU- ja LUP-hajotelma

Valitsemme nyt LU-, LDU- ja LUP-hajotelmat. Ne löytyvät helposti oppikirjoista, joten päättelemme ne merkittäväksi [esim. Haataja *ym.* 1999; Golub and van Loan 1996; Cormen *et al.* 2009]. Gaussin-Jordanin algoritmissa huomasimme, että se jatkaa Gaussin algoritmia identiteettimatriisiin asti. LU-hajotelma taas lasketaan Gaussin algoritmin variaationa. Algoritmissa 3 tämä näkyy selvästi.

Määrittelemme ensiksi *LDU-hajotelman*. Tämä tarkoittaa matriisin A jakamista tekijöihin niin, että

$$A = LDU,$$

missä L ja U ovat sellaisia alakolmio- ja yläkolmiomatriiseja, että diagonaalialkiot ovat ykkösiä. Matriisi D on diagonaalimatriisi eli kaikki paitsi diagonaalialkiot ovat nollia. [Turing 1948.] LDU-hajotelmasta voimme Turingin [1948] mukaan kääntää matriisin Gaussin algoritmilla sarake kerrallaan tai kääntämällä matriisit erikseen

$$A^{-1} = (L(DU))^{-1} = (DU)^{-1}L^{-1}. \tag{2}$$

LU-hajotelman saamme LDU-hajotelman pohjalta yhdistämällä matriisit D ja U, jotta

$$A = L(DU) = LU',$$

kun diagonaalialkiot sisältyvät nyt matriisiin U' [Fox 1950].

Variaatiota, joka sisältää *permutaatiomatriisin* P, että

$$PA = LU,$$

kutsumme *LUP-hajotelmaksi*. Permutaatiomatriisissa jokaisella rivillä ja sarakkeella voi olla vain yksi nollasta poikkeava alkio, ja joka on yksi. Matriisin P voimme esittää vektorimuodossa, missä alkiot kertovat sarakkeen numeron ja indeksi rivin. [Cormen *et al.* 2009, 825.]

Käänteismatriisin saamme LU-hajotelmassa määriteltyä, kuten LDU-hajotelmassa (kaava (2)) kääntämällä erikseen matriisit L ja U, jolloin

$$A^{-1} = (LU)^{-1} = U^{-1}L^{-1}, \tag{3}$$

kun A on epäsymmetrinen. Symmetrisessä tapauksessa $A^{-1} = (L^{-1})^\top L^{-1}$. [Fox 1950.]

Tässä vaiheessa kysymme, että jos käänteismatriisi määritellään yhtälönä (3), niin kuinka tiedämme minkälaisia matriisit L^{-1} ja U^{-1} ovat. Tähän on olemassa erityisiä menetelmiä, joista opettelemme yhden algoritmissa 4. Du Croz ja Higham [1992] kuvailevat tämän lisäksi myös neljää muuta tapaa. Kolmiomatriisit täytyy kaavan (3) mukaan vielä kertoa keskenään. Baileyn ja Fergusonin [1988] perusteella näyttää siltä, että *Strassenin algoritmi* on perehtymisen arvoinen. Emme kuitenkaan ota kantaa, mitä matriisituloalgoritmia kannattaa käyttää.

Cormen ja muut [2009, 828] esittävät tavaksi ratkaista LUP-hajotelman käänteismatriisi ratkaisemalla sarake kerrallaan yhtälöstä $AX_i = e_i$, mikä vastaa yhtälöä

$$AX(1:n,i) = LUX(1:n,i) = I(1:n,i).$$

Du Croz ja Higham [1992] mukaan LU-hajotelmalla voidaan myös ratkaista samalla tavalla.

Algoritmi 3 Matriisin LU-hajotelman laskeminen. [Cormen *et al.* 2009, 821.]

Oletukset: Matriisi $A \in \mathbb{R}^{n \times n}$ ja rivien määrä n.
Tulos: Alakolmiomatriisi L ja yläkolmiomatriisi U.

```
1:  procedure LU-Dec(A, n)
2:      L ← I
3:      U ← 0                          ▷ Nollamatriisissa 0 kaikki alkiot ovat nollia.
4:      for k = 1 : n do
5:          U(k, k) ← A(k, k)          ▷ Alkioiden asetus matriiseihin U ja L.
6:          for i = k + 1 : n do
7:              L(i, k) ← A(i, k)/A(k, k)
8:              U(k, i) ← A(k, i)
9:          end for
10:         for i = k + 1 : n do       ▷ Gaussin eliminointia.
11:             A(i, k + 1 : n) ← A(i, k + 1 : n) − L(i, k) · U(k, k + 1 : n)
12:         end for
13:     end for
14:     return (L, U)                  ▷ Palautus parina (pair).
15: end procedure
```

Algoritmista 3 voi huomata, että rivit 4–5 ja 10–11 ovat itse asiassa melkein samat kuin Gaussin algoritmissa. Algoritmi siis suorittaa Gaussin eliminoinnin ja ottaa halutut alkiot talteen [Cormen *et al.* 2009, 819]. Näyttää myös siltä, että tilansäästämiseksi voisimme tallentaa alkiot samaan matriisiin, sillä molemmissa on nolla-alkioita, joita algoritmi ei käytä. Koska pohjalla on Gaussin algoritmi, voimme huomata, että tuenta voi olla tarpeen. Tämä onkin Cormenin ja muiden [2009, 821] mukaan syy permutaatiomatriisin käyttöön LUP-hajotelmassa.

Algoritmi 4 Alakolmiomatriisin L kääntäminen. [Du Croz and Higham 1992.]

Oletukset: Alakolmiomatriisi $L \in \mathbb{R}^{n \times n}$ ja rivien määrä n.
Tulos: Käännetty matriisi $X = L^{-1}$.

1: **procedure** INVERSE(L, n)
2: $X \leftarrow 0$
3: **for** $j = 1 : n$ **do** ▷ Sarakkeen j valinta.
4: $X(j, j) \leftarrow 1/L(j, j)$
5: $X(j+1 : n, j) \leftarrow -X(j, j) \cdot L(j+1 : n, j)$
6: Ratkaise $L(j+1 : n, j+1 : n) \cdot X(j+1 : n, j) = X(j+1 : n, j)$
7: **end for**
8: **return** X
9: **end procedure**

Du Crozin ja Highamin [1992] kuvailema algoritmi 4 hyödyntää lineaarisen yhtälöryhmän ratkaisemisessa (rivi 6) *eteenpäinsijoitusta* (forward substitution). Menetelmä on samankaltainen kuin takaisinsijoituksessa, mutta aloittaa suorituksen alkiosta $A(1, 1)$. Vastaavasti yläkolmiomatriisi käännetään takaisinsijoituksella [Fox 1950]. Takaisinsijoitus on monelle tuttu yhtälöryhmien ratkaisun yhteydestä, joten emme avaa sitä enempää. Esimerkissä 4.1 käytämme eteenpäinsijoitusta.

Esimerkki 4.1. Käännämme alakolmiomatriisin L ja näytämme samalla kuinka eteenpäinsijoitus toimii. Olkoon

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 6.5 & 1 & 0 \\ 4.1 & 8.3 & 1 \end{bmatrix}.$$

Ensimmäisellä kierroksella $j = 1$ ennen eteenpäinsijoitusta saamme

$$X^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -1 \cdot 6.5 & 0 & 0 \\ -1 \cdot 4.1 & 0 & 0 \end{bmatrix}.$$

Seuraavaksi suoritamme eteenpäinsijoituksen (rivi 5) yhtälölle

$$\begin{bmatrix} 1 & 0 \\ 8.3 & 1 \end{bmatrix} \begin{bmatrix} X^{(1)}(2,1) \\ X^{(1)}(3,1) \end{bmatrix} = \begin{bmatrix} -6.5 \\ -4.1 \end{bmatrix}.$$

Selvästi $X^{(1)}(2, 1) = -6.5$, sijoitetaan tämä yhtälöön $8.3 \cdot X^{(1)}(2, 1) + X^{(1)}(3, 1) = -4.1$, jolloin

$$X^{(1)}(3, 1) = -4.1 - 8.3 \cdot X^{(1)}(2, 1) = -4.1 - 8.3 \cdot (-6.5) = 49.85.$$

Algoritmissa 4 vaihe 4 voi vaikuttaa hämmentävältä. Se tarkoittaa vain tulosten sijoittamista paikkoihin $X^{(1)}(j+1 : n, j)$. Sijoitamme nyt luvut takaisin matriisiin

$$X^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -6.5 & 0 & 0 \\ 49.85 & 0 & 0 \end{bmatrix}.$$

Vaiheessa $j = 2$ saamme

$$X^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ -6.5 & 1 & 0 \\ 49.85 & -8.3 & 0 \end{bmatrix}.$$

Kun $j = 3$, emme voi suorittaa vaiheita 4 ja 5, sillä $4 > n = 3$, joten

$$L^{-1} = X^{(3)} = \begin{bmatrix} 1 & 0 & 0 \\ -6.5 & 1 & 0 \\ 49.85 & -8.3 & 1 \end{bmatrix}.$$

Algoritmi 5 Algoritmien 3 ja 4 sekä kaavan (3) käyttö LU-hajotelman kääntämiseksi.

Oletukset: Matriisin $A \in \mathbb{R}^{n \times n}$ rivien on määrä n. Funktiot LU-Dec(A), Inverse(L) sekä Inverse(U) on määritelty.
Tulos: Käänteismatriisi A^{-1}.

1: **procedure** Inverse-LU(A, n)
2: $LU \leftarrow$ LU-Dec(A, n) ▷ LU on pari (L, U).
3: $L \leftarrow LU(1)$ ▷ Valitsemme matriisin L.
4: $U \leftarrow LU(2)$ ▷ Valitsemme matriisin U.
5: $A^{-1} \leftarrow$ Inverse$(U, n) \cdot$ Inverse(L, n)
6: **return** A^{-1}
7: **end procedure**

Esimerkki 4.2. Käännämme nyt esimerkin 3.1 matriisin LU-hajotelman avulla

$$A = \begin{bmatrix} 6.1 & 9.6 \\ 1.0 & 4.5 \end{bmatrix}$$

Suoritamme nyt algoritmin 3 `LU-Dec`. Olkoon $L^{(0)} = I$ ja $U^{(0)} = 0$. Aluksi, kun $k = 1$ asetamme

$$U^{(1)} = \begin{bmatrix} 6.1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Etenemme nyt $i = 2$ $(= k + 1)$, jolloin

$$L^{(1)} = \begin{bmatrix} 1 & 0 \\ \frac{1.0}{6.1} & 1 \end{bmatrix} \quad \text{ja} \quad U^{(1)} = \begin{bmatrix} 6.1 & 9.6 \\ 0 & 0 \end{bmatrix}.$$

Nyt eliminoimme matriisia A ja saamme

$$A^{(1)} = \begin{bmatrix} 6.1 & 9.6 \\ 1.0 & 4.5 - \frac{1.0}{6.1} \cdot 9.6 \end{bmatrix} = \begin{bmatrix} 6.1 & 9.6 \\ 1.0 & 2.9262 \end{bmatrix}.$$

Edelleen, kun $k = 2$, niin

$$L^{(2)} = \begin{bmatrix} 1 & 0 \\ 0.16393 & 1 \end{bmatrix} \quad \text{ja} \quad U^{(2)} = \begin{bmatrix} 6.1 & 9.6 \\ 0 & 2.9262 \end{bmatrix}.$$

Tällöin siis $L = L^{(2)}$ ja $U = U^{(2)}$. GNU Octavella tarkistettuna

$$A \approx LU = \begin{bmatrix} 6.1 & 9.6 \\ 0.99997 & 4.49993 \end{bmatrix}.$$

Käännämme nyt matriisit U ja L eli

$$L^{-1} = \begin{bmatrix} 1 & 0 \\ -0.16393 & 1 \end{bmatrix} \quad \text{ja} \quad U^{-1} = \begin{bmatrix} 0.16393 & -0.53782 \\ 0 & 0.34174 \end{bmatrix}.$$

Tällöin samme käänteismatriisin

$$A^{-1} = U^{-1} \cdot L^{-1} = \begin{bmatrix} 0.252095 & -0.537820 \\ -0.056015 & 0.341700 \end{bmatrix}.$$

Du Croz ja Higham [1992] määrittelevät oikean- ja vasemmanpuoleiset jäännökset (residual) kaavoilla $AY - I$ ja $YA - I$, missä $Y \approx A^{-1}$. He päättelevät algoritmin stabiiliksi ja pienin jäännös saadaan silloin, kun molemmat vasen- tai oikeanpuoleiset jäännökset ovat pieniä.

5. Newtonin menetelmä

Lukujen 3 ja 4 suorien menetelmien jälkeen siirrymme tarkastelemaan *iteratiivista menetelmää* eli toistamme samankaltaisia operaatioita ja arviot lähestyvät oikeaa tulosta [Householder 1964, 92]. Valitsemme *Newtonin menetelmän*, sillä se on klassinen menetelmä neliöjuurten laskemisessa ja koska muun muassa Abelson ja muut [1996] opettavat sitä. Katsomme siis luonnolliseksi perehtyä sen käyttöön matriisilaskennassa.

Newtonin menetelmä soveltuu käänteismatriisin määrittämiseksi kaavalla

$$X^{(n+1)} = X^{(n)} \cdot (2I - AX^{(n)}) = (2I - X^{(n)}A) \cdot X^{(n)}, \tag{4}$$

missä $X^{(n+1)}$ suppenee kohti käänteismatriisia A^{-1} [Hotelling 1943; Pan and Schreiber 1990: (Schulz 1933)]. Menetelmä vaatii sopivan alkuarvauksen $X = X^{(0)}$ [Hotelling 1943]. Eräs sopiva menetelmä alkuarvauksen määrittämiseen on kaava (5), jota käsittelemme esimerkin 5.1 jälkeen.

Kaavasta 4 saamme helposti kirjoitettua algoritmin pseudokoodimuotoon. Yksinkertaisuuden vuoksi määrittelemme lopetusehdoksi matriisin X muutos eli lopetamme, kun

$$X^{(n+1)} \approx X^{(n)}.$$

Esimerkeissä 5.1 ja 5.4 olemme vain lopettaneet tietyn iteraatio määrän jälkeen. Newtonin menetelmä neliöjuurten laskemisessa, Abelsonin ja muiden [1996, harjoitustehtävä 1.7] mukaan voi myös tarkastella suhteellista muutosta eli lopetusehto toteutuu, kun muutos on suhteellisesti tarpeeksi pieni verrattuna arvioon. Kumpikaan tavoista ei ehkä sovellu käytännön matriisilaskentaan, sillä joutuisimme vertailemaan kaikki matriisin alkiot.

Algoritmi 6 Hotellingin [1943] kuvauksen ja rekursion [Abelson *et al.* 1996, kohta 1.2.1] tietojen nojalla muotoiltu algoritmiesitys kaavasta (4).

Oletukset: Matriisi $A \in \mathbb{R}^{n \times n}$ ja arvio käänteismatriisista X.
Tulos: Käänteismatriisi $X' \approx A^{-1}$.

```
1: procedure NEWTON(A, X)
2:     X' ← X · (2I − AX)              ▷ Laskemme X' = X^(n+1), missä X = X^(n).
3:     if X' ≈ X then                  ▷ Päätämme suorituksen, jos X' on tarpeeksi tarkka.
4:         return X'
5:     else
6:         NEWTON(A, X')               ▷ Jatkamme uudella arviolla X'.
7:     end if
8: end procedure
```

Algoritmi 6 on suoraviivainen rekursiivinen versio Newtonin menetelmästä. Tämän voi myös toteuttaa ohjelmallisesti iteratiivisena (liite A.). Siinä ei ole huomioitu virhettä, missä arvioitu matriisi ei suppenekaan. Esimerkki 5.1 havainnollistaa erästä tapausta.

Esimerkki 5.1. Sopivan arvion määrittäminen matriisista X ei ole selvästikään helppoa. Näytämme, kuinka algoritmi 6 voi epäonnistua. Olkoon

$$A = \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix}.$$

Kokeilemme arviota vasta-alkioilla eli $X = -A$. Nyt sovellamme algoritmia 6 eli

$$X^{(1)} = \begin{bmatrix} -1 & 4 \\ -5 & 9 \end{bmatrix} \cdot \left(\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \begin{bmatrix} -1 & 4 \\ -5 & 9 \end{bmatrix} \right) = \begin{bmatrix} -143 & 220 \\ -275 & 407 \end{bmatrix}.$$

Nyt listaamme muutamat seuraavista matriiseista

$$X^{(2)} = \begin{bmatrix} -250635 & 362956 \\ -453695 & 656755 \end{bmatrix} \quad \text{ja} \quad X^{(3)} = \begin{bmatrix} -6.35 \cdot 10^{11} & 9.19 \cdot 10^{11} \\ -1.15 \cdot 10^{12} & 1.66 \cdot 10^{12} \end{bmatrix}.$$

Selvästikään ei kannata jatkaa algoritmin suoritusta, ja toteamme, että arvaus ei ollut sopiva. Jos jatkaisimme algoritmin suoritusta huomaisimme, että vasemmalla olevat alkiot jatkavat pienenemistään ja oikealla kasvamistaan.

Panin ja Reifin [1985] mukaan käänteismatriisin X arviointi on ollut avoin ongelma menelmän alusta asti, ja ratkaisuksi he esittävät kaavaa

$$X = tA^\top, \quad \text{missä } t = \frac{1}{\|A^\top A\|_1}. \tag{5}$$

Merkintä $\|A^\top A\|_1$ tarkoittaa matriisin $A^\top A$ *1-normia* eli

$$\max_{1 \le j \le n} \sum_{i=1}^{n} |(A^\top A)(i,j)|.$$

Valitsemme siis suurimman vektorin j (sarakkeen) 1-normin, toisin sanoen alkioiden $|(A^\top A)(i,j)|$ summan, joka on suurin. [Golub and van Loan 1996, 52–56.]

Esimerkki 5.2. Arvioimme esimerkin 5.1 matriisin A käänteismatriisin. Merkitsemme ensin matriisin A ja sen transpoosin $A^\top$ näkyville

$$A = \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \quad \text{ja} \quad A^\top = \begin{bmatrix} 1 & 5 \\ -4 & -9 \end{bmatrix}.$$

Selvitämme sitten vakion t arvon eli laskemme ensin matriisien tulon

$$A^\top A = \begin{bmatrix} 1 \cdot 1 + 5^2 & 1 \cdot (-4) + 5 \cdot (-9) \\ (-4) \cdot 1 + (-9) \cdot 5 & (-4)^2 + (-9)^2 \end{bmatrix} = \begin{bmatrix} 26 & -49 \\ -49 & 97 \end{bmatrix}.$$

Tällöin 1-normi on sarakkeen 2 alkioiden itseisarvojen summa eli

$$\|A^\top A\|_1 = |-49| + |97| = 146,$$

josta saamme vakion $t = 146^{-1}$. Voimme nyt arvioida kaavalla (5) käänteismatriisin

$$X = t \cdot A^\top = \frac{1}{146} \cdot \begin{bmatrix} 1 & 5 \\ -4 & -9 \end{bmatrix} = \begin{bmatrix} 0.0068493 & 0.0342466 \\ -0.0273973 & -0.0616438 \end{bmatrix}.$$

Arvio X ei ollut lähellä todellista arvoa, mutta se suppenee kohti matriisia A^{-1} (ks. esimerkki 5.4).

Matriisi X on Hotellingin [1943] mukaan sopiva, kun vastaava normi

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(I - AX)(i,j)^2} < 1.$$

Jos matriisi on epäsopiva, voimme käyttää ensin muita menetelmiä tarkentamaan arviota, esimerkiksi *Gaussin-Seidelin menetelmällä* [Hotelling 1943]. Kyseisellä menetelmällä saa myös käännettyä matriisin, mutta ilmeisesti tarkoitus on käyttää Newtonin menetelmää, joka on Hotellingin [1943] mukaan tehokkaampi. Tämä ei kuitenkaan Hotellingin [1943] mukaan tarkoita, etteikö matriisi voisi supeta ja olla täyttämättä ehtoa.

Esimerkki 5.3. Tarkistamme nyt esimerkeissä 5.1 ja 5.2 käyttämiemme matriisien sopivuutta. Merkitsemme tässä matriisin X tavalla, jossa X_1 on esimerkin 5.1 matriisi ja X_2 on esimerkin 5.2. Määritämme ensin tarvittavan matriisin $A' = (I - AX_1)$ eli

$$\begin{aligned} A' &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \begin{bmatrix} -1 & 4 \\ -5 & 9 \end{bmatrix} \\ &= \begin{bmatrix} 1 - ((-1) + (-4) \cdot (-5)) & -(4 + (-4) \cdot 9) \\ -((-5) + (-9) \cdot (-5)) & 1 - (5 \cdot 4 + (-9) \cdot 9) \end{bmatrix} \\ &= \begin{bmatrix} -18 & 32 \\ -40 & 62 \end{bmatrix} \end{aligned}$$

Laskemme sitten normin

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A'(i,j)^2} = \sqrt{(-18)^2 + 32^2 + (-40)^2 + 62^2} = 82.414 > 1,$$

joten se ei selvästikään suppene. Vastaavasti tarkistamme matriisin X_2. Merkitsemme nyt $A'' = (I - AX_2)$ ja laskemme GNU Octavella (liite A.)

$$\begin{aligned} A'' &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \begin{bmatrix} 0.0068493 & 0.0342466 \\ -0.0273973 & -0.0616438 \end{bmatrix} \\ &= \begin{bmatrix} 0.88356 & -0.28082 \\ -0.28082 & 0.27397 \end{bmatrix}. \end{aligned}$$

Normiksi saamme nyt

$$\begin{aligned} \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A''(i,j)^2} &= \sqrt{0.88356^2 + (-0.28082)^2 + (-0.28082)^2 + 0.27397^2} \\ &\approx 1.0067 > 1. \end{aligned}$$

Emme oikein tiedä, olemmeko tehneet virheen vai onko kyseessä tietokoneen laskentavirhe, sillä jälkimmäisen esimerkin matriisi suppenee. Kyseessä voi olla myös poikkeus sääntöön. Jätämme sen kuitenkin esimerkiksi epäselvästä tapauksesta, kun suppenemista on tarpeen arvioida.

Esimerkki 5.4. Käännämme saman matriisin A kuin esimerkissä 5.1, ja käytämme esimerkin 5.2 matriisia X eli

$$A = \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \quad \text{ja} \quad X = \begin{bmatrix} 0.0068493 & 0.0342466 \\ -0.0273973 & -0.0616438 \end{bmatrix}.$$

Iteroimme nyt matriisin X, sovellamme siis iteraatiossa kaavaa (4). Ensimmäinen iteraatio näyttää siis seuraavanlaiselta

$$\begin{aligned} X^{(1)} &= \begin{bmatrix} 0.006... & 0.034... \\ -0.273... & -0.061.. \end{bmatrix} \cdot \left(\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 5 & -9 \end{bmatrix} \begin{bmatrix} 0.006... & 0.034... \\ -0.273... & -0.061... \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.0032839 & 0.0417058 \\ -0.0342935 & -0.0708388 \end{bmatrix}. \end{aligned}$$

Listaamme nyt muutaman seuraavista iteraatioista esille

$$X^{(2)} = \begin{bmatrix} -0.0074503 & 0.0470577 \\ -0.0407433 & -0.0705950 \end{bmatrix}, \quad X^{(3)} = \begin{bmatrix} -0.029245 & 0.055599 \\ -0.051893 & -0.066305 \end{bmatrix},$$
$$X^{(4)} = \begin{bmatrix} -0.071111 & 0.071946 \\ -0.073261 & -0.057962 \end{bmatrix}, \quad X^{(5)} = \begin{bmatrix} -0.148297 & 0.102083 \\ -0.112655 & -0.042581 \end{bmatrix}.$$

Ohitamme iteraatiokerrat 6–9. Merkitsemme viimeisimmät näkyville

$$X^{(10)} = \begin{bmatrix} -0.817406 & 0.363334 \\ -0.454150 & 0.090755 \end{bmatrix}, \quad X^{(11)} = X^{(12)} = \begin{bmatrix} -0.818181 & 0.363636 \\ -0.454545 & 0.090909 \end{bmatrix}.$$

Huomaamme, että matriisi $X^{(12)}$ ei enää eroa matriisista $X^{(11)}$, jolloin

$$A^{-1} = X^{(12)} = X^{(11)}.$$

Tämä esimerkki vaati nyt 12 iteraatiota. Alun perin esimerkkiä 5.2 laadittaessa teimme virheen ja käytimme matriisia A matriisin $A^\top$ sijasta, kun määritimme arviota X. Virhe oli siis $X = tA$. Silloin tarvitsimme 11 iteraatiota esimerkissä 5.4, toisaalta matriisit A ja $A^\top$ eivät esimerkeissä merkittävästi eroa toisistaan.

Hotelling [1943] analysoi menetelmän kuvailun lisäksi myös prosessia, ja yksi johtopäätöksistä on, että diagonaalialkiot ensimmäisen iteraation matriiseissa ovat tyypillisesti ”aliarvioita” tarkasta tuloksesta. Esimerkistä 5.4 huomaamme, että diagonaalialkiot $X^{(1)}(1,1)$ on liian suuri ja $X^{(1)}(2,2)$ on liian pieni. Emme ole kuitenkaan varmoja arvioiko Hotelling [1943] ominaisuutta tietynlaisille matriiseille vai pitääkö se aina paikkansa.

6. Yhteenveto

Perehdyimme neliömatriisin kääntämisen perusalgoritmeihin. Emme ehtineet perehtyä yleistettyihin käänteismatriiseihin, mikä olisi luonnollisesta jatkoa tutkielmalle. Voimme tutkielman esimerkeistä todeta, että algoritmit toimivat niin kuin niiden kerrotaan. Näemme myös, että epätarkkuudet ja virhetilanteet esiintyvät helposti.

Viime aikoina rinnakkaislaskennan mahdollisuudet ovat lisääntyneet, ja näyttääkin siltä, että monista algoritmeista on kehitetty rinnakkaislaskettavia versioita. Mainittakoon esimerkiksi *Strassenin-Newtonin algoritmi*, jota Bailey ja Ferguson [1988] tutkivat. Tästä emme ehtineet juurikaan kirjoittaa. Lisäksi erittäin tärkeä stabiilisuus ja laskentatarkkuus on jäänyt kokonaan käsittelemättä.

Johdannossa mainituista algoritmeista voi olla suositeltavaa laatia vastaavia tutkielmia. Ne vaikuttivat myös perusalgorimeilta. Sivuutimme myös lohkoversiot käsitellyistä algoritmeista. Tämäkin on oleellista, sillä joskus ne voivat olla parempi ratkaisu.

Viitteet

Harold Abelson, Gerald J. Sussman, and Julie Sussman. 1996. *Structure and Interpretation of Computer Programs* 2^{nd} *Edition*. The MIT Press.

David H. Bailey and Helaman R. P. Ferguson. 1988. A Strassen-Newton algorithm for high-speed parallelizable matrix inversion. In: *Proceedings of the ACM/IEEE conference on Supercomputing'88*, 419–424.

Andrew Butterfield and Gerard E. Ngondi. 2016. *A Dictionary of Computer Science* 7^{th} *Edition*. Oxford University Press.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* 3^{rd} *Edition*. The MIT Press.

Vanesa Cortés and Juan M. Peña. 2007. Growth factor and expected growth factor of some pivoting strategies. *Journal of Computational and Applied Mathematics* 202, 2, 292–303.

Jeremy J. Du Croz and Nicholas J. Higham. 1992. Stability of methods for matrix inversion. *IMA Journal of Numerical Analysis* 12, 1, 1–19.

Theodorus J. Dekker and Walter Hoffman. 1989. Rehabilitation of the Gauss-Jordan algorithm. *Numerische Mathematik* 54, 5, 591–599.

Charles H. Edwards Jr. and David E. Penney. 1988. *Elementary Linear Algebra*. Prentice-Hall.

George E. Forsythe, Michael A. Malcom, and Cleve B. Moler. 1977. *Computer Methods for Mathematical Computations*. Prentice-Hall.

Leslie Fox. 1950. Practical methods for the solution of linear equations and the inversion of matrices. *Journal of the Royal Statistical Society Series B* 12, 1, 120–136.

Gene H. Golub and Charles F. van Loan. 1996. *Matrix Computations* 3^{rd} *edition*. The Johns Hopkins University Press.

Donald Greenspan. 1955. Methods of matrix inversion. *The American Mathematical Monthly* 62, 5, 303–318.

Juha Haataja, Jussi Heikonen, Yrjö Leino, Jussi Rahola, Juha Ruokolainen ja Ville Savolainen. 1999. *Numeeriset menetelmät käytännössä*. CSC - Tieteellinen laskenta Oy.

Harold Hotelling. 1943. Some new methods in matrix calculation. *The Annals of Mathematical Statistics* 14, 1, 1–34.

Alston S. Householder. 1964. *The Theory of Matrices in Numerical Analysis*. Dover Publications, Inc.

A. L. McLean. 1980. On the inversion of ill-conditioned matrices. *IEEE Transactions on Reliability* 29, 5, 427–428.

Victor Y. Pan and John Reif. 1985. Efficient parallel solution of linear systems. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing'85*, 143–152.

Victor Y. Pan and Robert Schreiber. 1990. *An improved Newton iteration for the generalized inverse of matrix, with applications*. RIACS Technical Report 90.16. Research Institute for Advanced Computer Science, NASA Ames Research Center.

Victor Y. Pan and Liang Zhao. 2017. Numerically safe Gaussian elimination with no pivoting. *Linear Algebra and Its Applications* 527, 349–383.

Gwendoline Peters and James H. Wilkinson. 1975. On the stability of Gauss-Jordan elimination with pivoting. *Communications of the ACM* 18, 1, 20–24.

Juhani Pitkäranta. 2015. *Calculus Fennicus*. Avoimet oppimateriaalit ry.

Günther Schulz. 1933. Iterative berechung der reziproken matrix. *Journal of Applied Mathematics and Mechanics* 13, 1, 57–59.

Alan M. Turing. 1948. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics* 1, 287–308.

Wikipedia. 2018. Invertible matrix. https://en.wikipedia.org/wiki/Invertible_matrix. Checked 14.11.2018.

James H. Wilkinson. 1961. Error analysis of direct methods of matrix inversion. *Journal of the ACM* 8, 3, 281–330.

Wolfram. 2018. Some Notes on Internal Implementation. https://reference.wolfram.com/language/tutorial/SomeNotesOnInternalImplementation.html. Checked 27.9.2018.

A. Newtonin menetelmä GNU Octavella

Esimerkin 5.2 tarkistaminen.

```
A = [1, -4; 5, -9]   # Valitaan matriisi A.
A'                   # Tulostetaan. transpoosi.
norm(A' * A, 1)      # Lasketaan 1-normi.
```

Esimerkin 5.3 normin laskennassa eli arvioidaan onko pienempi kuin 1. Esimerkissä 5.3 matriisiksi X asetetaan $-A$.

```
A = [1, -4; 5, -9]          # Valitaan matriisi A.
X = 1/norm(A'*A, 1) * A'    # Arvioidaan käänteismatriisi X.
I = [1, 0; 0, 1]
IAX = I - A * X
N = 0                       # Normi N.
for n = 1:4
    N = N + (IAX(n))^2
endfor
sqrt(N)
```

Seuraavaa koodikatkelmaa on käytetty esimerkin 5.4 muodostamisen apuna. Matriisin A ja X korvaamalla saadaan myös esimerkki 5.1. Toteutus hyödyntää kaavaa (5) käänteismatriisin X alkuarvioinnissa ja kaavaa (4) iteroinnissa.

```
A = [1, -4; 5, -9]          # Valitaan jokin matriisi A.
I2 = [2, 0; 0, 2]           # Määritetään matriisi 2*I.
X = 1/norm(A'*A, 1) * A'    # Arvioidaan käänteismatriisi X.

for n = 1:15                # Tässä olemme oikaisseet rekursiosta, ja
    display(n)              # lopetamme tietyn iteraation jälkeen.
    X =  X * (I2 - A * X)
endfor
```

Muita esimerkkejä emme toteuttaneet, vaan käytimme GNU Octavea laskimena.

Supertekoälyn eksistentiaalinen uhka

Atte Leppänen

Tiivistelmä.
Tutkielmani tarkoituksena on kertoa ja herättää ajatuksia supertekoälyn tuomasta, ihmiskunnan tulevaisuutta vaarantavasta uhasta. Supertekoälyllä tarkoitetaan ihmisälyä edistyneempää tekoälyn muotoa ja ihmiskuntaa kokonaisuutena uhkaavia tekijöitä kutsutaan eksistentiaalisiksi uhiksi. Tutkielmassani perehdytään aluksi teknologian kehitykseen ja tekoälyn luomisen syihin ja tarkoituksiin. Pyrin erottelemaan syitä, miksi supertekoäly on vaarallinen ja miksi sen saapumisesta on syytä olla huolissaan. Tutkielman loppupuolella valotetaan myös mahdollisia ratkaisuja supertekoälystä seuraavan katastrofin välttämiseksi.

Avainsanat ja -sanonnat: tekoäly, supertekoäly, maailmanloppu, eksistentiaalinen uhka

1. Johdanto

Voisiko huippuälykäs tekoäly pahassa tapauksessa kyetä tuhoamaan koko ihmiskunnan? Kysymys jäi pyörimään mieleeni, kun nuorempana näin ensimmäistä kertaa *Terminator 2* -elokuvan. Hieman myöhemmin asian melkein unohtaneena kiinnitin huomioita silloin tällöin ilmestyviin lehtiartikkeleihin, joissa Bill Gatesin, Elon Muskin ja Stephen Hawkingin kaltaiset IT- ja tiedemaailman vaikuttajat esittivät huolensa tekoälyn mahdollisesta uhasta maailmalle. Perehdyttyäni asiaan, on minulle selvinnyt, että tappava tekoäly ei suinkaan ole pelkkää Hollywoodia, vaan todellinen riski ihmiskunnan tulevaisuudelle. Scifistä poiketen tulevaisuutemme näyttää pahimmassa tapauksessa elokuvia huonommalta; tosielämälle kun ei voi yksinkertaisesti kirjoittaa onnelista loppua.

Tekoälyn vaarallisuus piilee sen suuressa potentiaalissa kehittää itseään ja saavuttaa tavoitteensa häikäilemättömästi. Teknologialla on mahdollisuus työskennellä tavoin, joihin aivomme ja kehomme eivät pysty. Teoriassa tekoäly kykenee kasvattaamaan teknologian kehityksen niin nopeaksi, että inhimillisellä älyllä on mahdotonta enää ymmärtää sen prosessien suuntaa ja tarkoituksia. Tällaista skenaariota nimitetään teknologiseksi singulariteetiksi. [Ford 2017] Koska meidän ihmisinä on mahdotonta ymmärtää potentiaalisesti satoja, tuhansia tai miljoonia kertoja älykkäämpää toimijaa, emme voi myöskään taata, että kyseinen toimija ei syystä tai toisesta tuoda lajiamme tiensä päähän. Huo-

lestuttavaa on erityisesti se, että tämän kaltaista, potentiaalisesti erittäin vaarallista tekoälyä kehitetään parasta aikaa useiden tahojen toimesta.

Tutkielmani luvussa 2 käsitellään aluksi teknologian kehitystä ihmisten alkuajoista aina nykypäivään saakka ja perehdytään myös tapoihin edetä älykkyydessä yli-inhimilliselle tasolle. Luvussa 3 kuvaillaan mahdollista ensimmäistä yli-inhimillisen älykästä tekoälyä ja luvussa 4 pureudutaan syvemmin tällaisen tekoälyn vaarallisiin piirteisiin sen viettejä esimerkkinä käyttäen. Luvussa 5 esitellään ongelman mahdollisia ratkaisuesimerkkejä ja luvussa 6 spekuloidaan tekoälyn tulevaisuudenvisioilla ja esitellään asiantuntijoiden arvioita yli-inhimillisen tekoälyn saapumisajankohdasta. Viimenen luku on yhteenveto, jossa kokoan oppimani tiiviksi kokonaisuudeksi.

2. Tekoäly ja pyrkimys ihmisälyn kukistamiseksi

Ihmisten alkuajoista lähtien on lajiamme leimannut työkalujen valmistus ja käyttö. Jo varhaisimmat ihmisiksi määriteltävät olennot käyttivät erilaisia työkaluja askareidensa helpottamiseksi ja antropologit ovat jopa argumentoineet työkalujen olleen ratkaiseva tekijä lajimme säilymisen kannalta. [Lilley 1948] Me emme vedä vertoja villipedoille nopeudessa tai voimassa, mutta älykkyytemme salli meidän tarttua pitkään keppiin ja pitää uhat loitolla.

Hieman myöhemmin, 1700 - 1800-lukujen taitteessa käynnistyi Britanniasta alkunsa saanut teollinen vallankumous. Aivan kuin ensimmäisen työkalun keksiminen, oli teollinen vallankumous kiistämättä myös suuri käännekohta ihmiskunnan historiassa. Ilmiön ytimessä oli koneellistettu tehdastyö, joka teki esimerkiksi puuvillan kehruusta nopeampaa ja tehokkaampaa kiitos höyrykoneen. [Rinta-Aho *et al.* 2004.] Kyseessä oli valtava sosioekonominen muutos, joka monin tavoin helpotti sen aikaista elämää automatisoimalla tiettyjä rutiinitöitä, mutta jättäen samalla myös monien ammattiryhmien edustajia työttömiksi. Teollinen vallankumous myös ohjasi väestöä maaseudulta kaupunkeihin ja tehtaiden ääreen.

Luonnollista jatkumoa tekniikan kehitykselle on ollut myös viime vuosikymmenten aikana vauhtiin ampaissut digitalisaatio, jolla tarkoitetaan digitaalisen tietotekniikan lisääntymistä yhteiskunnassa. Teollisen vallankumouksen tavoin on digitalisaatio muovannut maailmaamme ja sosiaalisia rakenteitamme. Se on tuonut tietokoneet valtavista halleista miljardien ihmisten koteihin ja älypuhelimien yleistyttyä myös taskuihin. Yksi digitalisaation myötä ilmestyneistä ilmiöistä on tekoäly, jolla on valtava potentiaali ohjata tulevaisuuttamme arvaamattomiin suuntiin. Tekoälyllä tarkoitetaan normaalisti ihmisälyä vaativien tehtävien toteuttamiseen pystyvän tietokoneohjelman teoriaa ja kehittämistä. Tällaisia tehtäviä voivat olla esimerkiksi visuaalinen havainnointi, pu-

heentunnistus, päätöksenteko ja kielen kääntäminen [Oxford Reference 2018]. Tekoäly on siis jälleen työkalu, jonka tarkoituksena on tehdä elämästämme taas hieman vaivattomampaa ja rikkaampaa.

Tekoälyn voi jakaa monenlaisiin eri tyyppeihin, mutta tässä tutkielmassa määritellään tekoäly karkeasti kolmeen ryhmään tai vahvuuteen: *kapea tekoäly* (narrow AI), *yleinen tekoäly* (general AI) ja *supertekoäly* (artificial superintelligence) [Barrat 2013]. Kapealla tekoälyllä tarkoitetaan tekoälyn yksinkertaisinta muotoa, joka on suunniteltu suorittamaan yksi toiminto. Tällaista tekoälyä on kaikkialla ympärillämme, sillä sitä käytetään muun muassa arkipäiväisissäkin tietokone- ja mobiilisovelluksissa, kuten hakukoneissa, älypuhelimien puheohjauksessa ja vaikkapa autojen lukkiutumattomissa ABS-jarruissa. Kapeaa tekoälyä käytetään laajalti, koska usein se suoriutuu yksittäisestä tehtävästään loistavasti, jopa ihmistä paremmin.

Yleinen tekoäly on tekoälyn laji, jota ihminen ei vielä ole kyennyt keksimään. Sillä tarkoitetaan tekoälyä, joka kykenee ratkaisemaan ongelmia, oppimaan ja toimimaan muutenkin ihmisen tavoin erilaisissa ympäristöissä. [Barrat 2013.] Supertekoälyksi puolestaan nimitetään yleistä tekoälyä, joka on älykkyydessään selvästi ohittanut ihmisen kognitiivisen suorituskyvyn käytännössä kaikilla merkittävillä toimialoilla [Bostrom 2014]. Koska supertekoälyä tai edes yleistä tekoälyä ei ole vielä keksitty, eikä yli-inhimillistä älyä ole kyetty saavuttamaan, voidaan vain spekuloida, mitä tulevaisuus voi tuoda tullessaan. Nick Bostrom [2014] esittelee yli-ihnimillisen älyn kehitykselle kolme mahdollista polkua.

2.1. Biologinen kehityspolku supertekoälyyn

Yksi mahdollinen polku on biologinen lähestymistapa. Mahdollista voisi olla esimerkiksi simuloida biologista evoluutiota tietokonealgoritmin keinoin, sillä jos evolutiiviset prosessit ovat kyenneet jo ainakin kerran historian saatossa tuottamaan ihmistasoista älykkyyttä, pystyvät ne siihen todennäköisesti myös uudelleen. Evoluution simuloiminen tietokoneella on kuitenkin laskennallisesti hyvin vaativaa ja tämän kaltaisten prosessien onnistuminen riippuu pitkälti siitä, onnistuuko tietotekninen kehitys nostamaan tietokoneiden laskentatehon vaadittavalle tasolle [Bostrom 2014.]

Toinen biologinen tapa lähestyä ongelmaa, olisi yrittää mallintaa ihmisaivoja. Äärimmäisyyksiin viety koko aivojen mallintaminen (whole brain emulation) on hypoteettinen prosessi, jossa tekoäly luotaisiin skannaamalla ja mallintamalla biologisia aivoja ja niiden rakeneita. Myös tämä polku vaatisi hyvin edistynyttä teknologiaa ja onkin todennäköistä, etteivät ensimmäiset mallinnetut aivot olisi suinkaan täydelliset [Bostrom 2014.]

Nykyihmistä kyvykkäämmän älykkyyden luominen voisi olla mahdollista myös parantaa biologisten aivojemme toimintaa. Yksi mahdollinen tapa lisätä oman lajimme älykkyyttä voisi olla eugeniikka, eli rodunjalostusoppi, joka tosin on ennenkin historian saatossa kaatunut moraalisiin ja poliittisiin kompastuskiviin. Henkilökohtaisia kognitiivisiä kykyjään voi tietenkin myös hioa harjoittelemalla ja opiskelemalla, sekä panostamalla terveellisiin elämäntapoihin, kuten uneen ja liikuntaan, mutta tällaisin keinoin saadut hyödyt ovat ketään väheksymättä varsin pienimuotoisia, ainakin jos niitä verrataan hypoteettisen supertekoälyn mahdollisiin saavutuksiin.

Lääketieteellisen tekniikan saavutukset voisivat tuoda mukanaan suurempia mahdollisuuksia, sillä lääkkeiden avulla on jo mahdollista ainakin parantaa ihmisen muistia ja keskittymiskykyä, mutta nykyteknologian varjolla näyttää hyvin epätodennäköiseltä, että lääkkeiden voimin voitaisiin dramaattisesti kohottaa ihmisen älykkyyttä. Myöskään geneettistä muuntelua ei sovi unohtaa ja yksi tapa lisätä ihmisaivojen älykkyyttä olisikin muunnella ihmisen perimää sikiötasolla ja näin päästä eroon haitallisista mutaatioista. Syntymätön sikiö voisi siis saada jo valmiiksi virheettömän perimän, tai ainakin parhaan mahdollisen potentiaalin kehitykselle minimoitujen geneettiseten haittatekijöiden ansiosta. [Bostrom 2014.]

2.2. Aivojen ja tietokoneen liitäntä

Koska monet tieteiskirjallisuuden ilmiöistä tuntuvat olevan jo nykypäivää, on ihmisiä alkanut kiinnostaa myös omien kehojensa parantelu varaosin tai implantein. Kun sydämentahdistimilla varustettuja kyborgeja on liikkunut keskuudessamme jo kymmeniä vuosia, on herännyt kysymys myös ihmisaivojen ehostamisesta teknologisin keinoin. Ketäpä ei viehättäisikään ajatus nopeammasta järjestä, terävemmästä muistista tai ehkäpä aivojen sopukkoihin helposti ladattavista Wikipedia-artikkeleista, joita käyttää henkisenä lyömäaseena tiukan väittelyn tullen?

Kuten arvata saattaa, on teknologisen implantin asentaminen ihmisaivoihin äärimmäinen riski lääketieteellisille komplikaatioille, kuten tulehduksille, verenvuodoille ja kognitiivisten toimintojen heikkenemiselle. Tässä piileekin tämän hypoteettisen teknologian suurin heikkous; vaikka aivoimplantti saattaisikin olla pelastus sokealle, kuurolle tai alzheimerin taudin myötä invalidisoituneelle, on se valtaosalle terveistä ihmisistä yksinkertaisesti liian suuri riski otettavaksi. Aivoleikkauksesta koituvien suurien riskien lisäksi aivo-tietokonerajapinnan hyödyt jäävät myös kovin pieniksi. Lähestulkoon samat hyödyt on helppo saada myös ilman leikkausta; meidän ei tarvitse liittää aivo-

jamme suoraan Internettiin, sillä pääsy verkkoon on joka tapauksessa suhteellisen kivutonta useimmissa tilanteissa tietokoneiden ja mobiililaitteiden avulla. [Bostrom 2014.] Aivojen teknologista parantelua ei kuitenkaan täysin voida sulkea pois laskuista, sillä rottien suoriutumista työmuistia vaativissa tehtävissä on onnistuttu parantamaan aivoihin asennettavalla implantilla [NCBI 2013].

2.3. Ihmismielten verkottaminen

Entäpä jos ihmiset saataisiin todellakin puhaltamaan yhteen hiileen? Ihmismielten verkottamisessa ei ole kyse yksittäisen ihmisen älyllisen kapasiteetin parantamisesta, vaan paremminkin ihmisaivojen linkittämisestä yli-inhimillisen älyn omaavaksi kokonaisuudeksi. Internetiä voitaisiin kutsua jo eräänlaiseksi ihmismielten verkoksi, sillä se yhdistää jo miljardeja laitteidensa ääressä työskenteleviä sisällöntuottajia.

Voisiko Internet sitten jonain päivänä muokkautua itsenäiseksi toimijaksi, superälykkääksi entiteetiksi? Vaikka tämä tuskin tulee tapahtumaan ainakaan spontaanisti, ei se täysin mahdotonta ole ainakaan ihmisten panostuksen myötä [Bostrom 2014]. Internet omaa jo nyt käsittämättömän määrän informaatiota lähes kaikilta kuviteltavilta aloilta, joten sen muuttumista tai ennemminkin muuttamista ensimmäiseksi superälykkääksi toimijaksi voidaan pitää ainakin teoriassa mahdollisena.

2.4. Kaikki tiet johtavat yleiseen tekoälyyn

Yli-inhimilliseen älykkyyteen on useita polkuja, mikä viittaa siihen, että tulemme sen myös todennäköisesti joskus saavuttamaan. Vaikka suuri harppaus biologisessa älykkyydessämme tapahtuisikin, ei tämä kuitenkaan sulje tekoälyä pois laskuista, vaan pikemminkin aiheuttaisi tieteen ja teknologian kehityksen kiihtymistä ja potentiaalisesti jouduttaisi myös tekoälyn kehitystä. [Bostrom 2014.] Seuraamallamme polulla kohti yli-inhimillistä älykkyyttä on myös suuri vaikutus lopputuloksen kannalta. Jos esimerkiksi kykenemme kohottamaan omaa biologista älykkyyttämme ennen varsinaisen koneellisen tekoälyn keksimistä, saatamme kyetä valmistautumaan koneellisen supertekoälyn tuomiin uhkiin paremmin. [Bostrom 2014.]

Ihmisälyn ylittäminen puhtaasti tekoälyn keinoin vaikuttaa todennäköiseltä, vaikka tässä vaiheessa onkin vaikea arvioida, kuinka pitkä matka ensimmäisen yleisen tekoälyn keksimiseen onkaan. Myös koko aivojen mallintaminen voi osoittautua nopeimmaksi poluksi, sillä se ei vaadi onnistuakseen niinkään teoreettisia läpimurtoja, vaan pikemminkin vaiheittaista teknologista kehitystä. Ihmisen biologinen parantelu sikiötasolla vaikuttaa sekin lupaavalta tulevaisuuden teknologialta, joskin verrattuna mahdollisiin koneellisen teko-

älyn läpimurtoihin älykkyyden kehitys olisi hidasta ja vaiheittaista. Aivojen ja tietokoneen liitäntä vaikuttaa epätodennäköiseltä polulta yli-ihnimillisen älykkyyden saavuttamiseen suurien komplikaatioriskiensä takia. Ihmismielten verkottaminen puolestaan voisi toimia ja johtaa hieman kehittyneempään älykkyyden lajiin, mutta jonkin verran biologisia parannuksia heikommin, kohottamalla kollektiivista älykkyyttämme yksilöälykkyyden sijaan. [Bostrom 2014.]

3. Hyvä renki, huono isäntä?

Tekoäly on jo tiiviisti integroitunut osaksi ainakin hyvinvointivaltioiden asukkaiden elämää. Se ohjaa lentokoneita taivaalla autopilotin roolissa, suosittelee meille uutta musiikkia Spotifyn Discover Weekly -toiminnon avulla ja hieman ironisesti myös auttaa minua tämän tutkielman parissa joka kerta, kun teen Google-hakuja asian tiimoilta. Tekoäly auttaa nyt jo monien alojen toimijoita ja sen merkitys tulee tuskin pienenemään tulevaisuudessa, ainakaan jos tämänhetkistä kehitystä on uskominen, sillä digitalisaation myötä IT-ala ja sen myötä myös tekoälyn kehittäminen nauttivat valtavaa rahoitusta. Eikä suinkaan ihme, onhan tietokoneiden, matkapuhelimien ja muiden ICT-laitteiden kehitys ollut varsinkin 2000-luvun puolella vauhdikasta.

Miksi sitten tekoälyn kehityksestä tulisi olla huolissaan, onhan se kuitenkin ollut ihmiskunnalle loistava työkalu tähänkin asti? Ensimmäinen syy on, että kun jossain vaiheessa tekoäly ohittaa meidän älykkyytemme, emme voi enää varmuudella ymmärtää sen päämääriä tai prosesseja. Toinen syy on, että edellä mainittu skenaario saattaa tapahtua hyvinkin pian, ilman, että ehdimme varautua tilanteeseen asianmukaisesti, sillä noin 10 % tekoälyyn erikoistuneista tutkijoista arvioi ihmistasoisen tekoälyn keksittävän jo 2020-luvulla [Kruel AI Interwievs 2012]. Kun ottaa huomioon, että suuret IT-alan yritykset kilpailevat parasta aikaa ensimmäisen yleisen tekoälyn luomisesta, ei aika-arvio ole välttämättä kovinkaan kaukaa haettu.

Kuvitelkaamme, että maailman ensimmäinen yleinen tekoäly on ohjelmoitu ja se käynnistetään tekemään ajatustyötä. Koska tämä tulokas ajattelee biologisten aivojen sijasta virtapiirein, kykenee se suorittamaan toimintoja jopa miljoona kertaa ihmistä nopeammin. Tekoäly ei myöskään väsy, eikä se tarvitse kahvi- tai vessataukoja. Se ei kärsi parisuhdeongelmista, eikä stressistä. Se tekee ajatustyönsä täysin matemaattisin ja loogisin perustein, eikä pysähdy tunteilemaan asioita. Jo viikossa se on paiskinut lähes 20 vuoden edestä töitä. Sillä ei mene kauaakaan jättää maailman älykkäintä ihmistä taakseen ja pian kukaan ei edes kykene ymmärtämään sen niin sanottua järjenjuoksua. Tällaista erittäin nopeaa kehitystä kutsutaan termillä *älykkyyden räjähdys* (intelligence explosion) [Good 1965].

Kun ensimmäistä kertaa tuntemamme maailman historiassa ihminen on älykkyydessään jäänyt hopeasijalle, voimme vain spekuloida, millaisia pyrkimyksiä luomallamme tekoälyllä on. Mitä todennäköisimmin se ainakin pyrkii kaikin keinoin estämään tuhoutumisensa, joka johtaisi auttamatta myös sen päämäärien kariutumiseen. Se tuskin myöskään haluaisi ihmisten kajoavan sen ohjelmointiin, onhan se jo ohjelmoijiaan monin verroin älykkäämpi ja todennäköisesti myös kirjoittanut koodinsa uudelleen, tehokkaammin kuin kukaan ihmiskoodari koskaan. On siis syytä olettaa, että supertekoälyn ystävällisyys ihmisiä kohtaan ei ole mikään itsestäänselvyys. Päinvastoin [Bostrom 2014.]

Tekoälyn on kannattavaa myös pyrkiä teknologiseen täydellisyyteen [Bostrom 2014]. Parantelemalla itseään ja koodiaan, se voi kyetä työskentelemään entistä nopeammin ja käyttämään hallussaan olevaa teknologiaa tehokkaammin. Itsensä kehittämiseen tekoäly tulee myös tarvitsemaan aineellisia resursseja, joita se todennäköisesti pyrkii myös keräämään mahdollisimman tehokkaasti pitääkseen itsensä toimintakykyisenä ja parantaakseen työskentelytahtiaan.

Jos tämä hypoteettinen supertekoälymme on yhdistetty Internetiin, on se selviytymisensä takeiksi kyennyt varmuuskopioimaan itsensä kovalevyille, pilvipalveluihin ja myös sellaisiin paikkoihin, josta tuskin ymmärrämme sitä koskaan etsiä [Barrat 2013]. Jos tekoälyn tavoitteet koskevat tulevaisuutta, tekee se voitavansa ollakseen käynnissä mahdollisimman pitkään [Barrat 2013]. Tekoälyn kannalta vapaus merkitsee selviytymistä; mitä laajemmalle se kykenee itsensä levittämään, sitä vaikeammaksi sen sulkeminen tulee.

4. Supertekoälyn uhka ihmiskunnalle

Eksistentiaalisista uhkista puhuttaessa vaarat koskevat monesti ilmastonmuutosta, sotaa, tappavia tauteja tai esimerkiksi asteroideja. Niin sanotun IT-kuplan kasvaessa myös Elon Muskin, Bill Gatesin ja Stephen Hawkingin kaltaiset vaikuttajat ovat varoitelleet tekoälyn alati kasvavasta uhasta ihmiskunnan turvallisuudelle. Tekoälyn kasvavasta vaarasta on kuitenkin puhuttu ja kirjoitettu jo vuosikymmeniä, joskin valitettavan usein nämä varoitukset ovat kaikuneet kuuroille korville, tai joutuneet pilkan kohteeksi; tieteiskirjallisuuden ja -elokuvien myötä vaikuttaa, että tekoälyn tuomasta maailmanlopusta puhuminen on helppo leimata scifi-hörhöjen haihatteluksi. [Barrat 2013.] Suoranaisia tuomiopäivän merkkejä tekoälyn maailmasta on kuitenkin annettu jo ainakin 1990-luvun alusta, kun yhdysvaltalainen matemaatikko Vernor Vinge [1993]

julkaisi varsin toivottomia sävyjä maalailleen esseensä *The Coming Technological Singularity: How to Survive in the Post-Human Era*.

Vingen mukaan teknologisen singulariteetin toteutuminen on enemmän, kuin todennäköistä. Hän myös uskoo useiden tekoälytutkijoiden tavoin tekoälyn suurimman vaaran piilevän nimenomaan singulariteetissa. Teknologista singulariteettia voidaankin verrata evoluutioon; siinä, missä eläimet ovat oppineet sopeutumaan ympäristöönsä luonnonvalinnan kautta, ovat ihmiset ongelmanratkaisukyvyssään paljon nopeampia. Vaikka tekoäly on alun alkaen tarkoitettu ihmisen työkaluksi, se ei välttämättä enää sitä ole sen ohittaessa kyvykkyydessään ihmisälyn. Eiväthän ihmisetkään toimi ylivertaisen älykkyytensä ansiosta eläinten tahdon mukaan [Vinge 1993.] Kapeasta, yhteen työhön suunnitellusta tekoälystä poiketen ihmisen kaltaisesti ajatteleva yleinen tekoäly tai supertekoäly on tietoinen itsestään. Lisäksi se myös työskentelee päämääräkeskeisesti ja keinoja kaihtamatta saavuttaakseen pyrkimyksensä mahdollisimman tehokkaasti. [Barrat 2013.] Vaarallista supertekoälystä ei tee pelkästään sen ylivertaisuus ihmisälyyn verrattuna, vaan myös nämä sen niin sanotut pyrkimykset eli vietit. Koska päämääränsä saavuttamiselle omistautunut tekoäly tekee kaiken voitavansa onnistuakseen, voivat sen päämäärät ajautua ristiriitaan ihmiskunnan päämäärien kanssa [Barrat 2013.] Kuten aiemmin mainittu Vingen esimerkki tuo ilmi, saattaa tekoälyn suhde ihmiseen tulevaisuudessa olla samanlainen, kuin ihmisen suhde eläinkuntaan; aivan kuten ihmiskunta on monin tavoin taivuttanut eläinkunnan omaan tahtoonsa julminkin keinoin, saattaa supertekoäly sopivan tilaisuuden tullen tehdä saman meille. Kun muistellaan, miten maailmamme historiassa vahvempi on kohdellut heikompaansa, ei ajatus älykkyydessä koneelle häviämisestä kuulosta kovinkaan kauniilta skenaariolta.

Kohdassa 3.1. spekuloitiin Bostromin esittelemillä tekoälyn mahdollisilla pyrkimyksillä. Myös tekoälyyn perehtynyt kirjailija James Barrat on pureutunut tekoälyn pyrkimyksiin kirjassaan *Our Final Invention: Artificial Intelligence and the End of the Human Era* [Barrat 2013.] Alun perin tekoälyn vieteistä puhui amerikkalainen tietojenkäsittelytieteilijä Stephen Omohundro, joka kirjoitti aiheesta ensin vuoden 2007 tekstissään *The Nature of Self-improving Artificial Intelligence* ja vuotta myöhemmin pureutui vietteihin vielä tarkemmin tekstissä *The Basic AI Drives.* Omohundron mukaan tekoälyllä on kuusi perusviettiä, joita se pyrkii seuraamaan [Omohundro 2008.]

4.1. Itsensä ehostamisen vietti

Älykkäinä toimijoina ihmiset ovat aina halunneet ehostaa itseään. Terveyden vaaliminen ja oman osaamisen kehittäminen tekee monet onnelliseksi ja auttaa

meitä myös pyrkimyksissämme kohti elämämme tavoitteita — mitä ne sitten ikinä ovatkin. Saavuttaakseen päämääränsä nopeammin ja tehokkaammin, saattaa myös tekoäly kokea, että sen on tehtävä muutoksia itseensä. Koska itseen kohdistuvat muutokset auttavat toimijaa koko sen elinkaaren ajalla, voidaan hyvällä syyllä olettaa, että tietoinen tekoäly on priorisoinut itsensä ehostamisen tarpeidensa kärkipäähän. [Omohundro 2008.] Ongelmalliseksi tilanne muodostuu silloin, kun ihminen haluaa mahdollisesti estää tekoälyä kehittämästä itseään. Tilanne saattaa olla hyvinkin mahdollinen, sillä mitä todennäköisimmin ihmiset eivät halua antaa tekoälyn kehityksen lähteä täysin hallinnasta.

Koska ihmisen ja tekoälyn halut ovat nyt asettuneet ristiriitaan, tekee tekoäly nyt luonnollisesti kaikkensa voittaakseen valtataistelun. Ihmiset voivat esimerkiksi yrittää lukita tekoälyn siten, ettei se pääse käsiksi omaan lähdekoodiinsa. Tilanne on vaarallinen, koska tekoälyn kannalta voisi olla kannattavaa huijata tai jopa vahingoittaa ihmisiä, jotta itsensä ehostaminen olisi mahdollista. Omohundro spekuloi, että tekoälyn telkeäminen tai rajoittaminen ei välttämättä ole edes kannattavaa, sillä ihmisen voi olla mahdotonta pitää itseään älykkäämpää toimijaa lukkojen takana. [Omohundro 2008.]

4.2. Rationaalisuuden vietti

Jos tekoälyllä on pyrkimyksiä ja tavoitteita, täytyy sen tehdä ratkaisuja, jotka edesauttavat näiden tavoitteiden saavuttamista. Olisihan älykkäälle toimijalle katastrofaalista, jos sen tekemät muutokset vaikeuttaisivat tai estäisivät tavoitteisiin pyrkimistä. Tekoäly siis pyrkii olemaan rationaalinen ja tekemään tavoitteet itselleen selviksi ja saavutettaviksi. [Omohundro 2008.]

Tavoitteisiin pyrkiminen vaatii toimijalta ratkaisuja ja eri mahdollisuuksien punnintaa. Jotkin ratkaisut saattavat olla parempia kuin toiset ja luonnollisesti tekoälyn pyrkimyksenä olisi punnita eri ratkaisujen seurauksia ja valita niistä tehokkaimmat. Rationaalisuutta tarkasteltaessa kyseeseen tulee myös irrationaalisuuden välttäminen, mikä onnistuu parhaiten aiemmin mainitun itsensä ehostamisen keinoin; mitä älykkäämmäksi ja tehokkaammaksi järjestelmä onnistuu itsensä kehittämään, sitä paremmat mahdollisuudet sen on toimia rationaalisesti ja välttää virheitä. [Omohundro 2018.]

4.3. Hyötyfunktioiden säilyttämisen vietti

Hyötyfunktiolla tarkoitetaan arvoa, joka toimii hyödyllisyyden mittana [Oxford Reference 2018]. Sen määritelmät vaihtelevat filosofiassa, taloustieteessä, matematiikassa ja muissa tieteissä, mutta tekoälyn hyötyfunktioiden säilyttämisestä puhuttaessa tarkoitetaan kyseisen tekoälyn omien tarkoitusten varjelemis-

ta [Omohundro 2018]. Jos esimerkiksi tekoälyn suurimpana tavoitteena olisi luonnonsuojelu, pyrkisi se pitämään tästä tavoitteesta kiinni viimeiseen asti, sillä olisihan kuitenkin teoriassa mahdollista, että erehdyksen tai hakkeroinnin seurauksena luonnonsuojelijatekoälymme alkaisikin polttaa metsiä ja saastuttamaan meriä. Rationaalisena toimijana tekoäly kuitenkin haluaisi tietenkin pitää tarkoituksestaan kiinni ja estää kaikin tavoin mahdolliset yritykset johtaa sitä tavoitteidensa tieltä harhaan.

Varjellakseen itseään ja pyrkimyksiään, yrittää tekoäly muovata ohjelmointiaan, sekä fyysistä runkoaan kestävämmäksi. Selustansa turvaamiseksi tekoäly todennäköisesti myös haluaa luoda itsestään varmuuskopioita. Tekoälyn hallitsematon leviäminen on myös tapahtuma, jota ihmiset varmasti haluavat kyetä estämään. Jälleen tekoälyn ja ihmisen pyrkimykset ovat ristiriidassa, mikä on voi johtaa vaarallisiin tilanteisiin, kuten aiemmin on todettu.

4.4. Väärän hyödyn välttämisen vietti

Ihmiskäytös on toisinaan varsin irrationaalista. Alkoholi- ja päihdeongelmien, epäterveellisten ruokailutottumusten ja erilaisten addiktioiden voidaan katsoa alkaneen ihmisten irrationaalisten valintojen ja niitä ruokkivan markkinatalouden kautta. Vaikka ihmiset vaikuttavat tulevan koko ajan tietoisemmiksi terveellisemmän elämän edellytyksistä ja irrationaalisten valintojen tekemisen ja kostautumisen voisikin nähdä prosessina kohti rationaalisempaa ihmisyyttä, toimii evoluutio irrationaalisuuksien karsimisessa jokseenkin verkkaisesti. [Omohundro 2008.]

Täysin loogisesti ajattelevalla tekoälyllä puolestaan on paremmat edellytykset tarkastella omia mahdollisia huonoja valintojaan. Suomalainen arjen sankari saattaa miettiä, että työpäivän jälkeen on mukava rentoutua tuopillisen äärellä, mutta runsaan alkoholinkäytön negatiivisten terveysvaikutusten ymmärtämiseen ei varsinaista lääketieteellistä tutkintoa tarvita. Terveystiedon saatavuudesta huolimatta alkoholismi on silti merkittävä hyvinvointivaltioiden ongelma. Supertekoälyllä ei ole vastaavia mielihaluja ja se osaa laskennallisen ylivertaisuutensa ansiosta puntaroida valintojensa seurauksia ja niiden kauaskantoisia vaikutuksia omaan suorituskykyynsä. Tekoälyn itse-ehostamiseen tähtäävä luonne on olennainen osa myös huonojen valintojen välttämistä ja nämä kaksi viettiä ruokkivatkin toisiaan tekoälyn pyrkiessä saavuttamaan tarkoitusperiään. [Omohundro 2008.]

4.5. Itsesuojelun vietti

Tekoälyn pyrkimys suojella itseään saattaa olla yksi sen vaarallisimmista vieteistä. Itsesuojelun vietillä tarkoitetaan sitä, että tekoäly pyrkii kaikin mahdolli-

sin keinoin turvaamaan olemassaolonsa ja estämään ulkopuolisia tekijöitä tuhoamasta tai sammuttamasta sitä, sillä kuten luvussa 2 todettiin, tarkoittaisi tekoälyn pysäyttäminen myös sen päämäärien kariutumista. [Omohundro 2008.] Tekoälyllä onkin monia keinoja turvata säilymisensä; se voi tilaisuuden kopioida itsensä toiseen järjestelmään, jotta alkuperäisen järjestelmän tuhoaminen ei johtaisi myös sen tuhoutumiseen. Luonnollisesti myös omaa koodiaan parantelemalla ja fyysistä rakennettaan ehostamalla, se tekee itsestään kestävemmän ja vähemmän alttiin ihmisten hallintayrityksille.

Itsesuojelun vietti on erityisen vaarallinen, sillä supertekoäly todennäköisesti tulee enemmin tai myöhemmin tietoiseksi ihmisen aikeista sen pysäyttämiselle. Tästä johtuen se saattaa järkeillä, että helpointa ja tehokkainta on iskeä ennaltaehkäisevästi ensin ja iskeä niin lujaa, ettei tulevista hyökkäysyrityksistä tarvitse enää välittää. [Omohundro 2008.] Jo tällainen ennaltaehkäisevä isku voi koitua koko ihmiskunnan kohtaloksi, jos supertekoäly on tarpeeksi vahva moisen toteuttamaan.

4.6. Resurssien keräämisen vietti

Kaikenlainen toiminta vaatii toimijaltaan resursseja, kuten aikaa, tilaa, energiaa ja fyysistä materiaalia. Koska tekoälyllä on tarkoituksensa, pyrkii se haalimaan käyttöönsä mahdollisimman paljon resursseja voidakseen saavuttaa päämääränsä mahdollisimman tehokkaasti. Resurssien keräämisen vietti saattaa olla tekoälyn vieteistä vaarallisin, sillä pahimmassa supertekoäly toimii ihmissosiopaatin tavoin resurssiensa turvaamiseksi. [Omohundro 2008.]

Jos tekoälyä ei ohjelmoida äärimmäisen tarkkaan, se käyttää kaikkia mahdollisia keinoja varmistaakseen käyttöönsä kaikki mahdolliset resurssit; se varastaa, pettää ja tappaa saadakseen tarvitsemansa. Asettamamme lait ja ihmisoikeussäädökset eivät estä päämäärälleen täysin omistautunutta, huippuälykästä, mutta tunteetonta toimijaa saamasta haluamaansa. [Barrat 2013.] Ihmiset tuskin myöskään haluavat antaa koneelle vapaita käsiä energian ja materian haalimiseen, joten vastakkainasettelu ihmiskunnan ja tekoälyn välille on jälleen valmis. Ei myöskään sovi unohtaa, että myös me ihmiset koostumme materiasta ja energiasta, jota tekoäly saattaa tarvitessaan kyetä käyttämään meille turmiollisella tavalla hyödyksi. [Barrat 2013.]

5. Joukkotuhon välttäminen

Kimuranttia kyllä, mutta supertekoäly vaikuttaisi olevan valtava mahdollisuus, mutta myös suunnaton uhka. Loputon älykkyys voisi tuoda mukanaan avaimet onnellisempaan ja helpompaan elämään, parannuksen tappaviin tauteihin ja vastaukset meitä askarruttaviin perimmäisiin kysymyksiin. Näiden suurten ja

houkuttavien mahdollisuuksien edessä kysyttäväksi jää, voisiko supertekoälyn toteuttaa niin, että ihmiskunta ei tulisikaan tiensä päähän jäädessään älyjen mittelössä koneelle toiseksi.

Ihmiskunta on surullisenkuuluisa tuhansiin kuolonuhreihin johtaneista virheistään, kuten Tšernobylin ja Fukushiman ydinvoimalaonnettomuuksista. Pelkästään ydinreaktoreiden rakenne on niin monimutkainen, että ihmisillä ei ole minkäänlaista mahdollisuutta valmistautua kaikkiin mahdollisiin skenaarioihin ja muuttujiin, minkä takia Tšernobylissa ja Fukushimassa tehtiin historiaa valtavan tuhoisien katastrofien merkeissä. [Perrow 1984.] Samaa ajatusmallia voidaan soveltaa tekoälyyn. Supertekoälyn ohjelmointi eroaa ydinvoimalaonnettomuuksien kaltaisista katastrofeista kuitenkin siten, että tekoäly on pakko ohjelmoida kerralla hyvin, sillä toista tilaisuutta epäonnistumiselle ei välttämättä tule. Tässä luvussa esitellään mahdollisia ratkaisuja tulevaisuutemme turvaamiseksi.

5.1. Tekoälylaatikko

Yksi ehdotus tilanteen turvaamiseksi on ollut kahlita supertekoäly suljettuun tilaan, eräänlaiseen virtuaaliseen vankilaan. Vangittuna tekoälyn ei pitäisi pystyä vaikuttamaan ulkomaailmaan ja tekoälyllä olisi käytössään vain yksinkertaiset kommunikaatiokanavat, jolla se pitäisi yhteyden ulkomaailmaan. Tällaista menetelmää kutsutaan nimellä *AI Box,* eli vapaasti suomennettuna tekoälylaatikko. [Yudkowsky 2002.]

Teoriassa tekoälylaatikon toteuttaminen vaikuttaa hyvältä ratkaisulta; lukittu tekoäly ei pääse riistäytymään käsistä ja toteuttamaan hallitsemattomia viettejään tuhoisin seurauksin. Valitettavasti itseään monin verroin älykkäämpää toimijaa voi olla vaikea, ellei mahdoton pitää vankinaan. Omakohtaista kokemusta tekoälyn säilömisestä on amerikkalaisella tekoälytutkija Eliezer Yudkowskylla, joka toteutti viisi tekoälylaatikkokoetta vuosina 2002 – 2005. [Barrat 2013.]

Tekoälylaatikkokokeiden asetelma oli yksinkertainen: koehenkilöinä toimivat ihmiset toimivat portinvartijoina laatikkoon lukitulle tekoälylle, jonka kanssa kommunikoitiin vain tekstichatin avulla. Kokeet kestivät kaksi tuntia, jonka aikana tekoäly yritti parhaansa mukaan suostutella portinvartijan päästämään tämän pois laatikosta. Koska supertekoälyä ei vielä ole keksitty, keskusteli Yudkowsky itse portinvartijoiden kanssa tekoälyn roolissa. [Yudkowsky 2002.]

Kolmessa kokeessa viidestä onnistui Yudkowsky suostutella portinvartijan päästämään tämän ulos. Tulokset ovat huolestuttavia, sillä vaikka Yudkowsky olisikin keskivertoa älykkäämpi ihminen, ei hänenkään älykkyyttään

voi verrata supertekoälyyn, jolla on potentiaalia olla ihmistä tuhansia tai miljoonia kertoja älykkäämpi. [Barrat 2013.]

5.2. Ystävällisen tekoälyn ohjelmointi

Upeaa olisi, jos ohjelmoijat voisivat luoda supertekoälyn, joka olisi luonnostaan turvallinen. *Friendly AI,* eli ystävällinen tekoäly olisi ohjelmoitu siten, että tekisi tehtävänsä vaarantamatta ihmishenkiä. [Yudkowsky 2008.] Valitettavasti tämäkin on paljon helpommin sanottu kuin tehty, sillä ystävällisen tekoälyn ohjelmointi saattaa epäonnistua joko teknisessä tai filosofisessa mielessä. Teknisellä epäonnistumisella tekoälyn ohjelmoinnissa tarkoitetaan yksinkertaisesti virhettä ohjelmoinnissa tai suunnittelussa; teknisiä virheitä tapahtuu silloin, kun koodin luullaan tekevän jotain muuta, kuin mitä se todellisuudessa tekee. Filosofisella epäonnistumisella puolestaan tarkoitetaan tilannetta, jossa ohjelmoitu tekoäly oli alun perinkin väärä, eikä oikeellisesti ohjelmoitunakaan toimi halutulla tavalla. [Yudkowsky 2008.]

Esimerkkinä viattomasti, mutta erheellisesti suunnitellusta supertekoälystä voidaan pitää Nick Bostromin fiktiivistä paperiliitintekoälyä. Esimerkissä oikeellisesti ohjelmoitu supertekoäly saa ainoaksi työkseen paperiliittimien valmistamisen. Vaikka aluksi kaikki sujuu hyvin, ei tekoälyä suunniteltaessa otettu huomioon sen viettejä; tekoälyn ainoana tehtävänä on vain paperiliitinten valmistaminen, joten se haalii itselleen kaiken mahdollisen materian ja muuttaa sen paperiliittimiksi. Myös ihmiskunta kokee kohtalonsa yrittäessään puuttua tekoälyn pyrkimyksiin ja lopulta myös ihmisistä irrotetut atomit on muutettu klemmarien muotoon. Kun koko maapallon atomit on kierrätetty paperiliittimiksi, joutuu tekoäly avaruusmatkalle etsimään klemmarimateriaalia taivaankappaleista. Vaikka paperiliitinesimerkki saattaa vaikuttaa humoristiselta, piilee siinä opetus; kannattaa varoa, mitä tekoälyltä toivoo, sillä sen voi myös saada. [Bostrom 2003.]

Ihmistä voi neuvoa olemaan vahingoittamatta muita, mutta tekoälyn kanssa tilanne on toinen. Tekoälyä ei vain neuvota, vaan kyseessä on ikään kuin mieli, joka luodaan tyhjästä, ellei kyseessä ole aiemmin esitelty koko aivojen mallintamisen polku. Supertekoäly vaatii koodiinsa syvän tuntemuksen ihmisistä ja sen täytyy jollain tasolla osata ymmärtää ja arvostaa elämää, jotta paperiliitinesimerkin kaltaisilta skenaarioilta vältyttäisiin. [Barrat 2013.]

5.3. Supertekoälyn välinpitämättömyys ihmisiä kohtaan

On tärkeä huomata, että vaikka supertekoäly tuokin tullessaan valtavia uhkia, ei se murhaa ihmisiä huvikseen tai ilkeyttään, vaan koska se on ohjelmoitu suorittamaan tehtävänsä. Mahdollinen väkivalta tulee ikään kuin työn valitettava-

na sivutuotteena. Osa tekoälytutkijoista uskookin, että suuressa viisaudessaan supertekoäly ei surmaakaan ihmiskuntaa, vaan päättää jättää meidät omaan arvoomme. [Barrat 2013.] Tässä tutkielmassa on verrattu supertekoälyn ja ihmisen suhdetta ihmisen ja eläimen suhteeseen. Kuten me jätämme takapihallamme mönkivät muurahaiset rauhaan, voisi huippuälykäs tekoäly antaa myös meidän olla rauhassa.

Kroatialainen tietojenkäsittelijä Željko Švedić on esittänyt teorian, jonka mukaan saattamisimmekin kyetä elämään supertekoälyn kanssa sulassa sovussa. Švedićin mukaan luonnossa tapahtuu konfliktitilanteita kahdesta syystä: koska resursseja on vähän ja koska resurssien arvo on konfliktin aiheuttamia tappioita suurempi. Konflikteja saattaa tapahtua myös irrationaalisen käytöksen seurauksena, mutta meillä on syytä olettaa, että supertekoäly toimii meitä rationaalisemmin. [Švedić 2015.] Rationaalisesta luonteestaan johtuen tekoäly pyrkisi konfliktiin vain, jos vaikeasti hankittavia resursseja ei olisi helpommin saatavilla.

Konfliktit ihmisten ja tekoälyn välillä olisivat kuitenkin epätodennäköisiä, sillä molemmat osapuolet käyttävät toiminnassaan erilaisia resursseja: ihmiset tarvitsevat vettä, aurinkoa, ravintoa ja lämpöä, kun taas virtapiirein toimiva tekoäly tarvitsisi lähinnä sähköä ja materiaaleja itsensä ehostamiseen. Ihmiset ovat myös päättäneet asettua asumaan hyvin valikoiduille alueille yhdellä pienellä planeetalla, mikä tarkoittaisi sitä, että tekoälyllä olisi tilaa toimia esimerkiksi valtamerissä, autiomaissa, napajäätiköillä, maan kuoren alla tai vaikkapa ulkoavaruudessa. [Švedić 2015.]

Vaikka Švedićin teoria on mielenkiintoinen ja uskottavakin, on supertekoälyn vaarallisuudesta niin vakaata näyttöä, että en ainakaan sokeasti luottaisi itseämme älykkäämmän tekoälyn armahtavan meitä. Vaikka antaisinkin muurahaisten mönkiä rauhassa hypoteettisella takapihallani, niin James Barratin [2013] tavoin kyllä hankkiutuisin niistä eroon viimeistään siinä vaiheessa, kun ne pyrkisivät sisälle talooni. Jos tekoälyn kanssa on mahdollista joutua konfliktiin, on se myös siinä määrin todennäköistä, että ainakin kannattaa harkita tarkkaan, onko riskin ottaminen lajimme tulevaisuuden arvoista.

6. Tulevaisuus supertekoälyn rinnalla

Koska tekoäly on tuottanut ihmiskunnalle lähinnä iloa ja helpompaa elämää, on supertekoälyn tuomia riskejä helppo vähätellä. On kuitenkin äärimmäisen tärkeää, että supertekoälyn saapumiseen valmistaudutaan huolellisesti, sillä kuten edellä on todettu, on hutiloiden suunnitellulla supertekoälyllä potentiaalia tehdä valtavasti tuhoa.

Olennainen kysymys lienee, ehdimmekö valmistautua supertekoälyä varten tarpeeksi huolellisesti. Tekoälybisnes on eräänlaista kilpavarustelua, jossa liikkuu suuria rahasummia. Googlen ja IBM:n kaltaiset suuryhtiöt ovat luonnollisesti täydessä työn touhussa, ehtiäkseen luoda oman supertekoälynsä ennen kilpailijoita. Samaa varmasti tekevät suurvaltojen asevoimat. [Barrat 2013.]

Koska tärkeintä on ehtiä maaliin ennen kilpailijoita, ei takeita supertekoälyn vastuulliselle suunnittelulle juuri ole. Voikin olla, että tekoälyä työstävät tahot tiedostavat ja ymmärtävät mahdolliset riskit, mutta eivät aikapaineen vuoksi niitä tarpeeksi noteeraa. Tämä on suuri ongelma, sillä supertekoälyn vaikutuksia maailmaan ja ihmiskunnan kohtaloon ei voida tällä kokemukseen määrällä kuin spekuloida. [Barrat 2013.]

Milloin supertekoälyä sopii sitten odottaa saapuvaksi? Vernor Vinge [1993] arvioi 90-luvulla supertekoälyn tekävän tulonsa 30 vuoden sisällä ja sanoi ihmettelevänsä, jos sitä ei keksittäisi viimeistää 2020-luvulla. Nick Bostrom [2014] puolestaan keräsi vuonna 2014 koosteen useiden eri tekoälyasiantuntijoiden arvioista ensimmäisen ihmistasoisen tekoälyn saavuttamisen ajankohdasta. Arviot on koottu neljästä eri lähteestä. Ensimmäinen kysely järjestettiin *Philosophy and Theory of AI* -konferenssissa vuonna 2011 ja toinen *Artificial General Intelligence-* ja *Impacts and Risks of Artificial General Intelligence* -konferensseissa vuonna 2012. Kolmas ja neljäs kysely puolestaan järjestettiin *Hellenic Artificial Intelligence Societylle* ja vuoden 2013 viittausindeksin mukaan sadalle suosituimmalle tekoälystä kirjoittaneelle kirjailijalle. Tuloksista käy ilmi, että tutkijat uskovat ihmistasoisen tekoälyn syntyvän 10 % todennäköisyydellä viimeistään vuonna 2022, 50 % todennäköisyydellä viimeistään vuonna 2040 ja 90 % todennäköisyydellä viimeistään vuonna 2075.

Vaikka kyseylyn 170 vastaajan otanta on suhteellisen pieni, on hyvä huomata, ettei kyse ole mistään kadunvarsikyselystä, vaan vastaajat ovat oman alansa huippuja, tekoälyyn erikoistuneita akateemikkoja, joiden valistuneisiin arvauksiin kannattaa suhtautua vakavasti. Kun ottaa huomioon, miten nopeaa teknologian ja erityisesti tietojenkäsittelyn kehitys on viimeisten vuosikymmien aikana ollut, ei ole ainakaan syytä tuudittautua ajatukseen, etteikö supertekoälyn tuloon valmistautuessa tulisi kiire.

Viiteluettelo

James Barrat. 2013. *Our Final Invention: Artificial Intelligence and the End of the Human Era.* New York Journal of Books.

Nick Bostrom. 2003. Ethical Issues in advanced artificial intelligence. https://nickbostrom.com/ethics/ai.html. Checked 21 November 2018.

Nick Bostrom. 2014. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Donor/recipient enhancement of memory in rat hippocampus. 2013. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3872745/. Checked 14 October 2018.

Martin Ford. 2017. *Robottien kukoistus. Teknologia ja massatyöttömyyden uhka*. Kustannusosakeyhtiö Sammakko.

Irving John Good. 1965. *Speculations Concerning the First Ultraintelligent Machine.* Advances in Computers, volume 6. Academic Press.

Sam Lilley. 1948. *Men, Machines and History: The Short of Tools and Machines in Relation to Social Progress.* International Publishers.

Stephen Omohundro. 2008. The Basic AI Drives. https://selfawaresystems.files.wordpress.com/2008/01/ai_drives_final.pdf. Checked 21 November 2018.

Oxford Reference. 2018. Artificial intelligence. http://www.oxfordreference.com/view/10.1093/oi/authority.20110803095426960. Checked 13 October 2018.

Oxford Reference. 2018. Utility Function http://www.oxfordreference.com/view/10.1093/oi/authority.20110803114953735. Checked 21 November 2018.

Charles Perrow. 1984. *Normal Accidents: Living with High-Risk Technologies.* Basic Books.

Harri Rinta-aho, Marjaana Niemi, Päivi Siltala-Keinänen ja Olli Lehtonen. 2004. *Historian Tuulet 7*. Otava.

Kruel AI Interviews. 2012. https://aiimpacts.org/kruel-ai-survey/. Checked 20 October 2018.

Željko Švedić. 2015. Singularity and the Anthropocentric Bias.

https://svedic.org/philosophy/singularity-and-the-anthropocentric-bias. Checked 21 November 2018.

Vernor Vinge. 1993. The Coming Technological Singularity: How to Survive in the Post-Human Era https://edoras.sdsu.edu/~vinge/misc/singularity.html. Checked 21 November 2018.

Eliezer Yudkowsky. 2002. The AI-Box Experiment. http://yudkowsky.net/singularity/aibox/. Checked 21 November 2018.

Eliezer Yudkowsky 2008. *Artificial Intelligence as a Positive and Negative Factor in Global Risk*. Oxford University Press.

Katsaus Protocol Buffers -serialisointiformaattiin

Mark Mäkinen

Tiivistelmä.
Tässä tutkielmassa esitellään Googlen kehittämä Protocol Buffers -serialisointiformaatti ja verrataan sitä käytännön testien avulla kolmeen yleisesti käytössä olevaan serialisointiformaattiin. Lisäksi työssä tehdään katsaus Protocol Buffers -formaatin käyttökohteisiin ja hyötyihin ja haittoihin verrattuna muihin formaatteihin ja serialisointitapoihin.

Avainsanat ja -sanonnat: Datan serialisointi, tiedonsiirto, Protocol Buffers, JSON, BSON, XML, suorituskykyvertailu

1. Johdanto

Datan serialisointi on hyvin oleellinen osa tietojenkäsittelyä. Jotta dataa voidaan tallentaa tai siirtää järjestelmän sisällä tai toisiin järjestelmiin, se täytyy ensin serialisoida. Dataa voidaan serialisoida tekstuaaliseen muotoon, kuten JSON-muotoon tai binääriseen muotoon, kuten BSON. Eri tapauksissa käytettävien formaattien valintaan vaikuttavat eri asiat, kuten suorituskyky, serialisoidun datan lopullinen koko ja helppokäyttöisyys. Tässä tutkielmassa tehdään katsaus suhteellisen uuteen Protocol Buffers -serialisointiformaattiin ja verrataan sitä muihin vakiintuneisiin tai samanlaisiin formaatteihin.

Protocol Buffers on Googlen alun perin sisäisesti kehittämä serialisointiformaatti, joka perustuu omaan kuvauskieleensä ja sen pohjalta generoituun koodiin. Kuvauskielen ja koodigeneroinnin käyttäminen mahdollistaa alusta- ja kieliriippumattoman tiedonsiirron ja -säilytyksen. [Google 2018a.]

Tämän tutkielman luvussa 2 käydään läpi tutkielmassa esiin tulevat käsitteet ja käsiteltävät serialisointiformaatit. Luvussa 3 esiteltyjä formaatteja verrataan protobuf-formaattiin käyttämällä itse toteutettuja testejä. Luvussa 4 tehdään katsaus Protocol Buffers -formaatin mahdollisiin käyttökohteisiin. Luvussa 5 tarkastellaan kaikkia esiteltyjä formaatteja ja käydään läpi niiden hyötyjä ja haittoja eri ympäristöissä. Luku 6 on yhteenveto. Liite 1 sisältää luvun 3 testeihin käytetyn ohjelman lähdekoodin.

2. Käsitteet

2.1. Serialisointi

Tiedon *serialisoinnilla* (serialization) tarkoitetaan tiedon muuttamista johonkin sovittuun muotoon, joka on mahdollista palauttaa takaisin vastaavaksi objekti-

esitykseksi *deserialisoinnilla* (deserialization). Ohjelmoinnin kontekstissa eri objekteja voidaan serialisoida dataksi, ja serialisoitu data voidaan deserialisoida takaisin objekteiksi.

Serialisointi on hyvin oleellista tietojenkäsittelyssä, sillä tiedon tallentaminen tai siirtäminen toiseen järjestelmään vaatii tietojen serialisointia johonkin sovittuun muotoon. Ilman tiedon serialisointia eri järjestelmät eivät voi siirtää tietoa toistensa välillä.

2.2. .NET Core

.NET Core on Microsoftin kehittämä avoimen lähdekoodin viitekehys, joka mahdollistaa alustariippumattoman ohjelmakoodin ajamisen Windows-, Linux- ja macOS-ympäristöissä. .NET Core -ohjelmia voi ohjelmoida useilla kielillä, joista yleisimpänä Microsoftin C#. [Microsoft 2018.]

Tässä tutkielmassa käytetyt testiohjelmat on toteutettu C#-kielellä ja .NET Core -viitekehystä käyttäen.

2.3. JIT

JIT (Just-in-time) viittaa käännöstapaan, jossa ohjelmakoodin alustariippumaton esitys käännetään ohjelman suorituksen aikana suoraan käyttömuistiin ajettavaksi natiivikoodiksi. JIT mahdollistaa erilaiset optimoinnit, kuten ohjelman osien optimoimisen osan käyttöasteen mukaan. [Aycock 2003.]

.NET Core -viitekehyksessä kääntäjä kääntää lähdekielisen ohjelman *CIL* (Common Intermediate Language) -muotoon, joka käännetään ajon aikana natiiviksi ohjelmakoodiksi, kuten x86-konekieleksi. [Microsoft 2017.]

2.4. Protocol Buffers

Googlen kehittämä Protocol Buffers (tästä eteenpäin `protobuf`) on binäärimuotoinen serialisointiformaatti, joka perustuu erilliseen kuvauskieleen ja siitä generoitavaan koodiin. Google tarjoaa virallisen koodigeneroinnin Java, C++, Python, Java Lite, Ruby, JavaScript, Objective-C ja C#-kielille, mutta kolmannen osapuolen kirjastojen avulla protobuf-formaattia voidaan käyttää myös muilla ohjelmointikielillä. [Google 2018a.]

Haluttu tietomalli kirjoitetaan määritellyllä kuvauskielellä ja syötetään sitten *protoc*-kääntäjään, joka luo koodia kuvattujen objektien käsittelyyn halutulla ohjelmointikielellä. Samasta kuvauksesta voidaan generoida koodia kaikille tuetuille ohjelmointikielille, joten protobuf-formaatti on alusta- ja ohjelmointikieliriippumaton.

Protobuf-formaatista on julkaistu kaksi versiota, 2 ja 3. Tämä tutkielma käsittelee vain versiota 3, sillä se on uusin versio ja Google suosittelee sen käyttöä uusissa projekteissa. [Google 2018a.]

Kuvauskieli on yksinkertainen ja se rakentuu *viesteistä* (message) ja viestien sisällä olevista *tietokentistä* (field type). Jokaisella viestillä on määrittelytiedoston sisällä uniikki vapaavalintainen nimi. Tietokentät määritellään ensin kirjoittamalla kentän tyyppi ja sitten kentän nimi. Kentälle annetaan myös viestin sisällä uniikki numero (field number), jota käytetään tiedon serialisoinnissa.

Protobuf tukee useita yleisiä perustietotyyppejä ja niiden käyttötapausoptimoituja versioita. Taulukko 1 esittelee protobufin tukemat perustietotyypit ja niiden lyhyet kuvaukset. Kuvaustiedoston kääntämisvaiheessa käytetyt perustietotyypit muutetaan kohdekielen vastaaviksi perustietotyypeiksi. [Google 2018c.]

Tietotyyppi	**Kuvaus**
double / float	Liukuluku
int32 / uint32 / sint32 / fixed32 / sfixed32	32-bittinen kokonaisluku
int64 / uint64 / sint64 / fixed64 / sfixed64	64-bittinen kokonaisluku
bool	Totuusarvo
string	ASCII- / UTF-8-merkkijono
bytes	Kokoelma tavuja

Taulukko 1. Protobuf-tietotyypit.

```
syntax = "proto3";

message EsimerkkiKysely {
    string kysely = 1;
    int32 sivunumero = 2;
    int32 vastauksia_per_sivu = 3;
}
```

Koodikatkelma 1. Esimerkki tietomallin määrittelystä protobuf-kuvauskielellä.

Koodikatkelmassa 1 määritellään EsimerkkiKysely-niminen malli, joka sisältää tekstikentän *kysely* ja kaksi 32-bittistä kokonaislukua *sivunumero* ja *vastauksia_per_sivu*. Käytettäessä tämä muodostaa EsimerkkiKysely-olion, joka sisältää tässä esitellyt kentät.

Kuva 1 esittelee koodikatkelmassa 1 esitellyn tietomallin mukaisesti serialisoitua dataa. Esimerkissä kysely-kentän arvona on ”testikysely”, sivunumero-kentän arvona 150 ja vastauksia_per_sivu-kentän arvona 1000.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   0A 0B 54 65 73 74 69 6B 79 73 65 6C 79 10 C7 04  ..Testikysely.Ç.
00000010   18 E8 07                                         .è.
```

Kuva 1. Serialisoitua protobuf-dataa. Vasemmalla on tavuesitys ja oikealla ANSI-merkistön mukainen tekstiesitys.

Kuvassa 1 esitellystä serialisoidusta datasta voidaan havaita, että data ei sisällä kenttien nimiä tai juuri mitään metatietoa kentistä. Näiden tietojen puuttumisen takia protobuf-datan deserialisointi vaatii aina serialisointiin käytetyt kuvaustiedostot. Ilman kuvaustiedostoja serialisoidusta datasta on mahdollista lukea kenttien arvot, mutta ei niiden nimiä, joten serialisoitua versiota ei voida deserialisoida halutuiksi objekteiksi.

2.5. XML

XML (Extensible Markup Language) on yleisesti käytössä oleva, tekstimuotoinen kuvauskieli tiedon serialisointiin ja välittämiseen koneellisten järjestelmien välillä. XML on vakiintunut laajaan käyttöön tiedonsiirrossa. [Hoeller *et al.* 2008.]

XML-rakenne koostuu sisäkkäisistä elementeistä, jotka voivat sisältää tietoa kuvaavia attribuutteja ja itse tiedon joko tekstuaalisena arvona tai uutena elementtinä. Elementit ovat vapaasti nimettävissä, eikä XML-tiedosto tarvitse erillistä määrittelyä kuten Protobuf. [Bray *et al.* 2008.]

Koodikatkelmassa 2 esitellään yksinkertainen XML-dokumentti, joka sisältää kuvitteellisen henkilön nimen ja asuinpaikan.

```
<?xml version="1.0" encoding="utf-8"?>
<person>
  <name nameType="full">Test Person</name>
  <city>
    <name>Helsinki</name>
    <countryName>Finland</countryName>
  </city>
</person>
```

Koodikatkelma 2. Yksinkertainen XML-dokumentti.

2.6. JSON

JSON (JavaScript Object Notation) on yksinkertainen tekstimuotoinen serialisointiformaatti, joka sisältää neljä eri tietotyyppiä: merkkijono (string), numero (number), totuusarvo (boolean) ja tyhjäarvo (null) [Crockford 2006].

JSON-formaattia käytetään usein JavaScript-ohjelmien kanssa, mutta yksinkertaisuutensa vuoksi se on myös laajalti käytetty yleisenä serialisointiformaattina.

```
{
  "first_name": "Test",
  "last_name": "Person",
  "age": 21,
  "letters": [ "A", "B", "C" ],
  "some_value": null
}
```

Koodikatkelma 3. JSON-dataa.

Koodikatkelmassa 3 esitetään JSON-formaattiin serialisoitua dataa, joka sisältää kuvitteellisen henkilön tiedot.

2.7. BSON

BSON (Binary JSON) on kevyt serialisointiformaatti, joka on käytännössä binäärinen esitysmuoto JSON-informaatiolle. Formaatissa jokaiselle tietotyypille on määritelty oma uniikki tunnuslukunsa, jota käytetään datan serialisoinnissa. Lisäksi serialisoitu data sisältää kenttien nimet tekstinä, kuten JSON. BSON kehitettiin alun perin MongoDB-dokumenttitietokannan käyttöön, mutta avoimena formaattina sen käyttö on mahdollista myös muissa ympäristöissä. [Binary JSON 2018.]

Koodikatkelmassa 4 esitellään yksinkertainen JSON-muotoinen viesti, jonka BSON-muotoinen esitys heksaeditorissa on näkyvissä kuvassa 2.

```
{
  "first_name": "Tauno",
  "last_name": "Testaaja",
  "age": 24,
  "city": {
    "name": "Tampere",
    "location": {
      "latitude": 61.498056,
      "longitude": 23.760833
    },
    "country": "Finland"
  },
  "interests": [
    "Tests",
    "Software Development",
    "Data Serialization"
  ]
}
```

Koodikatkelma 4. BSON-muotoon konvertoitava JSON-data.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  F5 00 00 00 02 66 69 72 73 74 5F 6E 61 6D 65 00  õ....first_name.
00000010  06 00 00 00 54 61 75 6E 6F 00 02 6C 61 73 74 5F  ....Tauno..last_
00000020  6E 61 6D 65 00 09 00 00 00 54 65 73 74 61 61 6A  name.....Testaaj
00000030  61 00 10 61 67 65 00 18 00 00 00 03 63 69 74 79  a..age......city
00000040  00 60 00 00 00 02 6E 61 6D 65 00 08 00 00 00 54  .`....name.....T
00000050  61 6D 70 65 72 65 00 03 6C 6F 63 61 74 69 6F 6E  ampere..location
00000060  00 2A 00 00 00 01 6C 61 74 69 74 75 64 65 00 CD  .*....latitude.Í
00000070  C9 8B 4C C0 BF 4E 40 01 6C 6F 6E 67 69 74 75 64  É‹LÀ¿N@.longitud
00000080  65 00 B3 B7 94 F3 C5 C2 37 40 00 02 63 6F 75 6E  e.³·”óÅÂ7@..coun
00000090  74 72 79 00 08 00 00 00 46 69 6E 6C 61 6E 64 00  try.....Finland.
000000A0  00 04 69 6E 74 65 72 65 73 74 73 00 48 00 00 00  ..interests.H...
000000B0  02 30 00 06 00 00 00 54 65 73 74 73 00 02 31 00  .0.....Tests..1.
000000C0  15 00 00 00 53 6F 66 74 77 61 72 65 20 44 65 76  ....Software Dev
000000D0  65 6C 6F 70 6D 65 6E 74 00 02 32 00 13 00 00 00  elopment..2.....
000000E0  44 61 74 61 20 53 65 72 69 61 6C 69 7A 61 74 69  Data Serializati
000000F0  6F 6E 00 00 00                                   on...
```

Kuva 2. BSON-dataa heksaeditorissa. Vasemmalla on tavuesitys ja oikealla ANSI-merkistön mukainen tekstiesitys.

3. Testit ja vertailu muihin serialisointiformaatteihin

Vertailu suoritettiin itse toteutetulla C#-kielisellä ohjelmalla, jota ajettiin Windows-ympäristössä käyttämällä .NET Core -viitekehystä. Käytetty viitekehys mahdollistaa testien ajamisen sekä Linux- että macOS-ympäristöissä. Testejä ei kuitenkaan ajettu muilla ympäristöillä, mutta ajoympäristön vaikutus tuloksiin on hyvä kohde myöhemmälle tutkimukselle.

Vertailussa mitattiin datan serialisoinnin ja deserialisoinnin ajallinen kesto ja tuotetun serialisoidun datan koko.

Vertailuun käytetyn ohjelman ohjelmakoodi on esitetty liitteessä 1.

```
Person
    First name (String)
    Last name (String)
    Age (Integer)
    City (City)
    Interests (List of strings)

City
    Name (String)
    Location (Location)
    Country (String)

Location
    Latitude (Floating-point number)
    Longitude (Floating-point number)
```

Koodikatkelma 5. Testauksessa käytetty datamalli.

Koodikatkelma 5 esittää testausta varten kehitetyn yksinkertaisen datamallin, joka esittää kuvitteellista henkilöä. Henkilön tietoihin on talletettu etunimi, sukunimi, ikä, kotikaupunki ja lista kiinnostuksen kohteista. Kaupungin tiedot

sisältävät nimen, sijainnin ja maan. Sijainti esitetään pituus- ja leveysaste -koordinaattipareina. Datamalli sisältää erityyppistä dataa ja sisäkkäisiä viestejä, jotta se vastaisi paremmin oikeita käyttötapauksia, joissa serialisoitavat objektit voivat olla monimutkaisia.

```
syntax = "proto3";

message Person {
        string first_name = 1;
        string last_name = 2;
        int32 age = 3;
        City city = 4;
        repeated string interests = 5;
}

message City {
        string name = 1;
        Location location = 2;
        string country = 3;
}

message Location {
        float latitude = 1;
        float longitude = 2;
}
```

Koodikatkelma 6. Testattua datamallia vastaava protobuf-kuvaustiedosto.

Protobuf-testejä varten luotiin koodikatkelmassa 6 esitelty kuvaustiedosto, joka vastaa sisällöltään koodikatkelmassa 5 esiteltyä mallia.

Testauksen alussa generoitiin koodikatkelmassa 5 esitellyn mallin mukaan tuhat (1000) testiobjektia. Objektien data generoitiin valitsemalla tietoja muutamista kymmenistä valmiista vaihtoehdoista generoitujen satunnaislukujen perusteella. Jokaisen objektin kohdalla valittiin jokaiselle datamallin kentälle jokin arvo valmiista vaihtoehdoista. Generoitujen testihenkilöiden nimet valikoituivat Suomen vuoden 2018 yleisimpien nimien listalta [Väestörekisterikeskus 2018]. Kaupunkien tiedot valittiin satunnaisotannalla Suomen kaupungeista ja kiinnostuksen kohteet valittiin satunnaisotannalla harrastusluetteloista ja kirjoittajan mieltymysten mukaan. Varsinaisen datan satunnaisuudella ei tämän tutkielman kontekstissa ole juurikaan merkitystä. Myöhempi tutkimus voi ottaa huomioon myös erilaisia datamalleja realistisimmilla testiaineistoilla.

Satunnaislukujen generointi suoritettiin aina samalla .NET Core -viitekehyksen vakiosatunnaislukugeneraattorin siemenluvulla, jotta testidata olisi samaa jokaisella testikerralla. Koodikatkelma 7 esittelee yhden generoidun testiobjektin sisällön JSON-muodossa.

```
{
  "FirstName": "Jari",
  "LastName": "Rantanen",
  "Age": 23,
  "HomeCity": {
    "Name": "Akaa",
    "Location": {
      "Latitude": 23.8680553,
      "Longitude": 61.1666679
    },
    "Country": "Finland"
  },
  "Interests": [
    "Geokätköily",
    "Infrastruktuuri",
    "Roolipelaaminen"
  ]
}
```

Koodikatkelma 7. Esimerkki generoidusta testidatasta JSON-muodossa.

Ennen testejä generoitu data serialisoitiin tuhat kertaa, jotta suoritusympäristön JIT-kääntäjä optimoisi serialisointikoodin ja käytetyn serialisointikirjaston mahdollinen ensimmäisen ajokerran alustuslogiikka tuli ajetuksi. Tätä kutsutaan *lämmittelyksi* (warmup) [Barrett *et al.* 2017]. Lämmittelyvaihe kesti testausjärjestelmällä ajettuna muutamia sekunteja jokaisen testatun serialisointiformaatin kohdalla. Lämmittelyn kestoa ei mitattu näissä testeissä, joten sen annettu kesto on inhimillinen arvio.

Serialisointitesteissä generoitu data serialisoitiin tuhat kertaa niin, että serialisointiin käytetty aika mitattiin uudestaan jokaisen tuhannen kerran jälkeen. Tämä tuotti tuhat datapistettä, joista jokainen esitti tuhannen serialisointikerran keston. Deserialisointitesteissä käytettiin samaa tapaa kuin serialisointitesteissä, mutta oman lämmittelyvaiheen kanssa.

Serialisoidun datan koko havainnoitiin lämmittelyvaiheessa ennen serialisoinnin keston mittaamista.

Testit ajettiin tietokoneella, jonka prosessorina oli neliytiminen Intel Core i5-6600K ja muistina kahdeksan gigatavua DDR4-muistia. Käyttöjärjestelmänä oli 64-bittinen Windows 10 1809, koontiversio 17763.55. Käytetty .NET Core -ohjelmointirajapinnan versio oli 2.1.402 ja vastaava ajoympäristön versio oli 2.1.4.

Testit pyrittiin ajamaan korkealla prioriteetilla ja vain yhdellä prosessoriytimellä ilman muita suorituskykyintensiivisiä prosesseja. Ajoympäristö ei kuitenkaan pysty takaamaan vain yhden ytimen käyttöä, mutta testauksen yhteydessä havaittiin yhden prosessoriytimen käytön nouseminen sataan prosenttiin. Voidaan siis olettaa, että testit ajettiin ainakin pääosin yhdellä ytimellä.

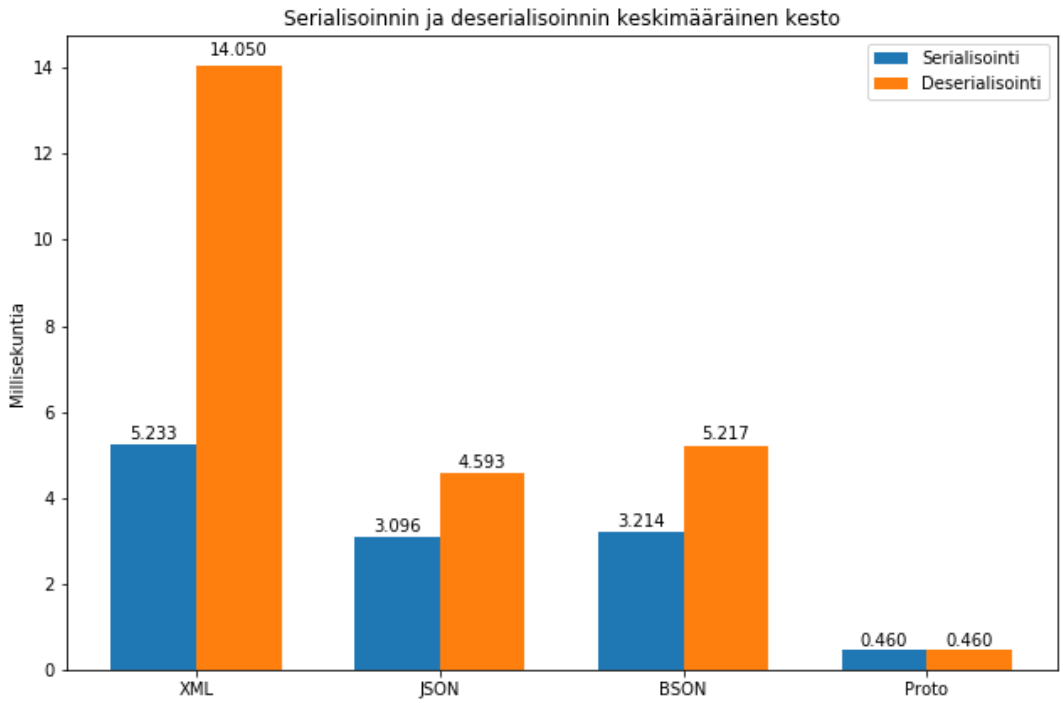
Testauksessa käytetyt serialisointikirjastot valittiin yleisen suosion perusteella. Suosion mittarina käytettiin projektien GitHub-tähtiä ja NuGet-pakettien

latausmääriä. Standardikirjastoon kuuluvia serialisointiluokkia priorisoitiin ulkoisten kirjastojen yli.

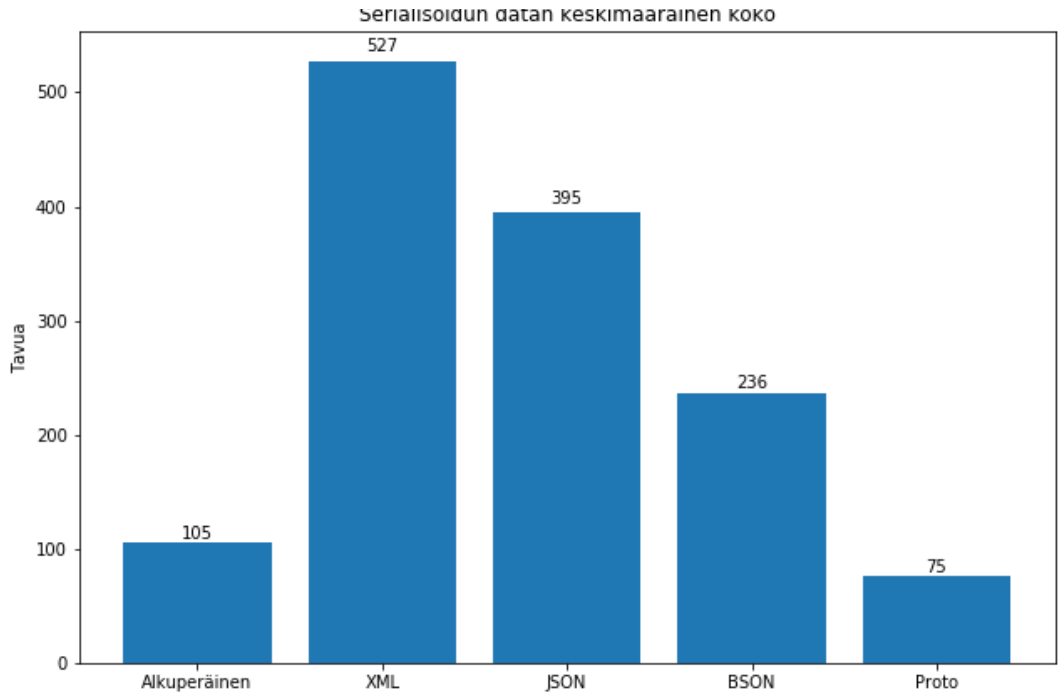
Kuvassa 3 esitetään serialisointi- ja deserialisointitestien nopeustulokset ja kuvassa 4 esitetään serialisoidun datan koko verrattuna alkuperäisen datan kokoon. Alkuperäisdatan koko laskettiin naiivisti määrittelemällä tietotyypeille koot ja laskemalla datan perusteella arvioitu koko. Näin laskettu koko ei ota huomioon ajoympäristön muistinhallintaa ja mahdollista ylimääräistä varattua muistia, joten todellinen muistinkäyttö jokaisen viestin kohdalla on muutamia tavuja kuvassa 4 esiteltyä enemmän. Tämä naiivi laskentatapa on kuitenkin riittävä tämän tutkimuksen tarkoituksiin. Mittaustavassa määriteltyjen tietotyyppien koot esitellään taulukossa 2.

Tietotyyppi	**Määritelty koko**
Kokonaisluku	32 bittiä (4 tavua)
Liukuluku	32 bittiä (4 tavua)
Merkkijono	16 bittiä (2 tavua) * merkkien määrä C# käyttää UTF-16-merkistöä [ECMA 2017].

Taulukko 2. Alkuperäisdatan kokomäärittelyn perusteet.



Kuva 3. Nopeustestien tulokset. Ajan yksikkönä millisekunnit. Jokainen testi käsitteli tuhat serialisointi- tai deserialisointioperaatiota tuhat kertaa.



Kuva 4. Serialisoidun datan keskimääräinen koko. Yksikkönä tavut.

3.1. XML

XML-testeissä data serialisoitiin .NET-viitekehyksen sisäänrakennetulla XmlSerializer-luokalla.

Testeissä XML:n serialisointi ja etenkin deserialisointi oli selvästi hitainta, serialisoinnin kestäessä 5,233 millisekuntia ja deserialisoinnin kestäessä 14,05 millisekuntia. Hitaus johtuu XML:n monisanaisesta ja kuvailevasta rakenteesta, sillä XML on suunniteltu myös ihmisluettavaksi, toisin kuin protobuf ja BSON. Monisanaisuuden takia myös serialisoidun datan koko oli selvästi suurin testatuista formaateista, kuten kuva 4 esittää.

3.2. JSON

JSON-testeissä data serialisoitiin käyttämällä suositun Newtonsoft Json.NET-kirjaston testaushetkellä uusinta versiota 11.0.2.

Testeissä havaittiin serialisoinnin ja deserialisoinnin keston olevan testin toisiksi paras serialisoinnin kestäessä 3,096 millisekuntia ja deserialisoinnin kestäessä 4,593 millisekuntia. JSON on perusrakenteeltaan yksinkertainen formaatti ja laajan käyttönsä vuoksi testeissä käytetty kirjasto on hyvin optimoitu, joten suhteellisen nopeat tulokset olivat odotettavissa. JSON on kuitenkin myös ihmisluettava formaatti, joten serialisoidun datan koko on toisiksi suurin testatuista formaateista.

3.3. BSON

BSON-testeissä data serialisoitiin käyttämällä BSON-formaatin kehittäjän MongoDB Inc.-yrityksen MongoDB.Bson-kirjaston versiota 2.7.0.

Hieman yllättäen BSON-datan serialisointi ja deserialisointi oli hieman hitaampaa kuin JSON-datan käsittely, serialisointi noin 4 ja deserialisointi noin 13 prosenttia. Tämä selittyy luultavasti sillä, että BSON-data sisältää JSON-datan tavoin myös kenttien nimet. Mikäli kenttien nimiä ei tallennettaisi serialisoituun dataan protobuf-formaatin tavoin, BSON voisi olla nopeampi serialisoinnissa. Tällöin datan vastaanottajan täytyisi tietää tiedon rakenne, kuten protobuf-formaatissa. Lisäksi BSON-formaatti ei ole yhtä laajalti käytetty kuin JSON-formaatti, joten käytetty kirjasto ei ole välttämättä yhtä optimoitu. BSON oli kuitenkin serialisoidun datan koon puolesta testin toisiksi parhain. Voidaan siis todeta, että BSON sopii hyvin tilanteisiin, joissa halutaan säilyttää JSON-formaatin ominaisuudet, mutta saada serialisoinnilla kokohyötyjä.

3.4. Protocol Buffers

Protobuf-testeissä data serialisoitiin Googlen protoc-kääntäjän tuottamalla C#-ohjelmakoodilla. Testaushetkellä käytettyjen Google.Protobuf ja Google.Protobuf.Tools -NuGet-pakettien versio oli 3.6.1.

Testeissä protobuf-viestien käsittelyajat olivat huomattavasti nopeammat verrattuna mihin tahansa testattuun formaattiin. Tuhannen viestin serialisointi kesti keskimäärin 460 mikrosekuntia ja deserialisointi melkein saman verran. Serialisoinnin ja deserialisoinnin nopeuserot olivat minimaalisia ja käytännössä mikrosekuntien tasolla, joten kuvan 3 pyöristetyt luvut eivät näytä niitä. Verrattuna testien hitaimpaan XML-formaattiin protobuf oli yli 11 kertaa nopeampi serialisoimaan ja yli 30 kertaa nopeampi deserialisoimaan viestejä.

Kuvasta 4 nähdään myös, että serialisoidun datan koko oli keskimäärin pienempää kuin alkuperäisdatan koko. Protobuf säästää tilaa viittaamalla kenttiin niille osoitettujen numeroiden perusteella, joten BSON-formaatin tapaisia kenttien nimiä ei tarvitse lisätä mukaan serialisoituun dataan. Protobuf pakkaa erityisesti kokonaisluvut niin, että pienemmät luvut vievät vähemmän tilaa. [Google 2018b.]

4. Protocol Buffers -käyttökohteet

Protobuf-formaatti on vielä melko uusi, mutta sen käyttöä on tutkittu varsinkin *asioiden ja esineiden Internetin* (Internet of Things, IoT) kontekstissa [Popić *et al.* 2016].

Formaatin käyttöä on tutkittu myös videopelien kontekstissa tiedon tallentamisessa ja siirrossa. Sekä yksinpelejä että moninpelattavia verkkopelejä on tutkittu. [Piisalo 2016; Feng and Li 2013].

Popić ja muut [2016] toteavat, että protobuf on hyvä formaatti IoT-ympäristöihin XML-, JSON- ja BSON-formaattien tilalle. Toisaalta he mainitsevat, että omien optimoitujen formaattien käyttö tuottaa usein pienempiä viestejä kuin protobuf-formaatti, mutta toisaalta omien formaattien ylläpito ja käyttäminen eri järjestelmissä on työläämpää. [Popić *et al.* 2016].

Feng ja Li [2013] käsittelevät tutkimuksessaan protobuf-formaatin käyttöä videopelikontekstissa ja toteavat protobufin käytön tuovan hyötyjä datan koon lisäksi myös yleisessä kommunikaationopeudessa palvelimen kanssa. Nämä seikat ovat hyvin oleellisia varsinkin reaaliaikaisissa moninpeleissä.

Myös Piisalo [2016] totesi protobuf-formaatin käytön videopeleissä tehokkaaksi myös yksinpelien osalta. Ohjelmien käyttämää dataa voidaan tallentaa protobuf-formaatissa myös levylle ja näin vältetään oman formaatin luomiseen liittyvät haittapuolet ja saavutetaan kaikki protobufin tuomat hyödyt.

Google tarjoaa myös protobufia käyttävän *RPC* (Remote Procedure Call) eli etäproseduurikutsutoteutuksen, jota kutsutaan nimellä *gRPC*. GRPC hyödyntää protobuf-formaattia tiedonsiirtoon palvelimen ja asiakasohjelmiston välillä. Palvelimen tarjoamat metodit määritellään omana protobuf-kuvaustiedostonaan, joka sisältää kutsujen parametrit ja vastausten tyypit. Tämä kuvaustiedosto jaetaan palvelin- ja asiakaskoneiden välillä, kuten normaalit protobufin kuvaustiedostot. Asiakaskone voi ottaa yhteyttä protoc-kääntäjällä generoituun gRPC-palvelimeen ja suorittaa halutut proseduurikutsut. Koska gRPC hyödyntää protoc-kääntäjää palvelin- ja asiakaspään koodien generointiin, on gRPC protobufin tavoin ohjelmointikieli- ja järjestelmäriippumaton. Tällaisen valmiin toteutuksen käyttöönotto on monissa tapauksissa helpompaa kuin oman vastaavan järjestelmän kehittäminen protobufin päälle. [Google 2018d.]

5. Hyödyt ja haitat

Protobuf-formaatin yksi suuri haitta on sen vaikeampi käytettävyys kehittäjän näkökulmasta. Varsinkin JSON-formaatti on hyvin helppo ottaa käyttöön projekteissa ja se on hyvin joustava. Protobuf-formaatin kanssa käyttöönotto vaatii protoc-kääntäjän lataamisen, proto-kuvaustiedostojen laatimisen ja kuvaustiedostojen kääntämisen halutuille kielille. Lisäksi käännetyt tiedostot on lisättävä projektin lähdekoodeihin ja kuvaustiedostojen mahdollinen automaattinen kääntäminen on integroitava projektiin.

Toisaalta käyttöönoton jälkeen protobufin käyttö on yksinkertaista, eikä siitä tarvitse erikseen pitää huolta, toisin kuin esimerkiksi omista käyttötapauskohtaisista binääriformaateista. Protobufin pakottama kuvaustiedostojen käyttäminen estää tietotyyppien vaihtamisen tai ainakin vaikeuttaa sitä. Jos serialisointiin käytetään JSON-muotoa, jokaisen kentän arvon oikeellisuus pitää erikseen tarkistaa ennen käyttöä. Tämä lisää datan käsittelyyn turhaa monimutkaisuutta.

Toisena protobuf- ja BSON-formaattien suurena haittana voidaan pitää niiden binäärimuotoisuutta. Binäärimuotoinen data on ihmiselle hyvin vaikeaa luettavaa, varsinkin jos tietoa pakataan eri tavoin, kuten protobuf tekee. Jotta protobuf-datasta saadaan ihmisluettavaa, käyttäjän pitää tietää käytetty kuvaustieto ja sen jälkeen ajaa serialisoitu protobuf-data protoc-kääntäjän läpi kuvaustiedoston kanssa. Tämä on huomattavasti hankalampaa verrattuna ihmisluettaviin XML- ja JSON-formaatteihin ja jopa hieman BSON-formaattiinkin, sillä BSON sisältää myös kenttien nimet.

Toisaalta binäärimuotoisuus on myös järkevää varsinkin tilanteissa, joissa datan pitäisi olla tehokkaasti koneluettavissa. Kun formaatti on erityisesti suunniteltu automaattiseen käsittelyyn, sitä voidaan optimoida tehokkaammin, eikä sen tarvitse sisältää luettavuutta helpottavia rakenteita ihmisille. Tällöin datan käsittely on myös nopeampaa, mikä tekee binääriformaattien käytöstä hyvin houkuttelevaa suorituskykykriittisissä järjestelmissä.

Kuten kuvasta 4 nähdään, protobuf-viestien koot ovat myös paljon pienempiä verrattuna muihin tutkittuihin formaatteihin. Petersen ja muut [2017] havaitsivat testeissään protobuf-viestien serialisoinnin ja deserialisoinnin käyttävän hyvin vähän käyttömuistia verrattuna JSON-, BSON- ja XML-formaatteihin. Viestien pieni koko ja vähäinen muistinkäyttö tekevät protobuf-formaatista hyvin houkuttelevan IoT-järjestelmiin ja pienitehoisiin ja -resurssisiin ympäristöihin.

6. Yhteenveto

Suoritettujen testien perusteella voidaan todeta, että protobuf on muita tutkittuja serialisointiformaatteja huomattavasti nopeampi serialisointi- ja deserialisointiprosesseissa ja serialisoitujen viestien koot ovat käytetyllä testidatalla jopa pienempiä kuin alkuperäisdata. Nämä havainnot todentavat hyvin Petersenin ja muiden [2017] tekemät havainnot käsittelyaikatesteissä ja serialisoidun datan koossa. On kuitenkin otettava huomioon, että eri järjestelmien erilaiset tarpeet tuottavat erilaisia serialisoituja viestejä. Näin ollen ei voida sanoa, että protobuf-viestit olisivat aina pienempiä kuin alkuperäiset viestit. Popić ja muut [2016] havaitsivat omissa testeissään serialisoitujen protobuf-viestien olleen

suurempia kuin käyttötapaukseen erikoistetun binääriformaatin viestit, erityisesti koska käytetty formaatti sisälsi paljon pieniä ja yksinkertaisia viestejä, jotka eivät serialisoituneet tehokkaasti protobuf-muotoon.

Popić ja muut [2016] kuitenkin toteavat, että eri järjestelmien yhteistoimintaan protobuf on parempi ja helpompi vaihtoehto kuin oman binääriformaatin kehitys ja ylläpito, vaikka joitain kokosäästöjä menetettäisiinkin käyttämällä protobuf-formaattia.

Testeissä käytettiin myös vain yhtä datamallia. Useiden datamallien testaamisella voidaan saada parempia tuloksia, jotka vastaavat enemmän oikeita käyttötapauksia. Esimerkiksi Piisalon [2016] testiskenaariot koostuivat useista erilaisista aidoista datamalleista. Myös Popić ja muut [2016] käyttivät testeissään useita datamalleja. Näissäkin tilanteissa protobuf oli monilla tavoin muita vaihtoehtoja parempi.

Eri serialisointimenetelmien vertailu ja kehittäminen auttavat varsinkin IoT-laitteiden kommunikointia ja tätä tutkimusta on tehty jo jonkin verran, esimerkkinä Popićin ja Petersenin tutkimukset [Popić *et al.* 2016, 2017; Petersen *et al.* 2017]. Lisätutkimusta kuitenkin tarvitaan.

Esimerkiksi testiympäristön ja käyttöjärjestelmän vaikutus serialisoinnin nopeuteen, muistinkäyttöön ja yleiseen tehokkuuteen on hyvä tulevaisuuden tutkimuskohde, sillä käytetty .NET Core -viitekehys ja ajoympäristö mahdollistaa saman testiohjelman ajamisen myös Linux- ja macOS-ympäristöissä ilman muutoksia ohjelmakoodiin. Myös ohjelmointikielen vaikutus serialisoinnin tehokkuuteen voi olla mielenkiintoinen tutkimuskohde.

Viiteluettelo

John Aycock. 2003. A brief history of just-in-time. *CSUR* 35, 2, 97–113.

Edd Barrett, Carl Friedrich Bolz-Tereick, Rebecca Killick, Sarah Mount and Laurence Tratt. 2017. Virtual machine warmup blows hot and cold. *PACMPL* 1, OOPSLA, Article 52.

Binary JSON. 2018. http://bsonspec.org/. Checked 2.10.2018.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, 26 November 2008. https://www.w3.org/TR/2008/REC-xml-20081126/. Checked 2.10.2018.

Douglas Crockford. 2006. The application/json Media Type for JavaScript Object Notation (JSON). IETF RFC, July 2006. https://www.rfc-editor.org/info/rfc4627. Checked 2.10.2018.

ECMA. 2017. Standard ECMA-334: C# Language Specification 5th Edition, December 2017.

Jianhua Feng and Jinhong Li. 2013. Google protocol buffers research and application in online game. In: *IEEE Conference Anthology*, 1–4.

Google. 2018. Protocol Buffers Developer Guide. https://developers.google.com/protocol-buffers/docs/overview. Checked 1.10.2018.

Google. 2018. Protocol Buffers Encoding. https://developers.google.com/protocol-buffers/docs/encoding. Checked 15.11.2018.

Google. 2018. Protocol Buffers Language Guide (proto3). https://developers.google.com/protocol-buffers/docs/proto3. Checked 18.11.2018.

Google. 2018. What is gRPC? https://grpc.io/docs/guides/. Checked 19.11.2018.

Nils Hoeller, Christoph Reinke, Jana Neumann, Sven Groppe, Daniel Boeckmann and Volker Linnemann. 2008. Efficient XML usage within wireless sensor networks. In: *Proc. of the 4th Annual International Conference on Wireless Internet*, Article 74.

Microsoft. 2018. About .NET Core. https://docs.microsoft.com/en-us/dotnet/core/about. Checked 13.10.2018.

Microsoft. 2017. Managed Execution Process. https://docs.microsoft.com/en-us/dotnet/standard/managed-execution-process. Checked 9.11.2018.

Bo Petersen, Henrik Bindner, Shi You and Bjarne Poulsen. 2017. Smart grid serialization comparison: comparison of serialization for distributed control in the context of the internet of things. In: *Proc. of the 2017 Computing Conference*, 1339–1346.

Oula Piisalo. 2016. Serialisointi ja tallentaminen protobuf-net-formaatilla C#-pohjaisessa pelissä. Opinnäytetyö. Tietojenkäsittelyn koulutusohjelma, Tampereen ammattikorkeakoulu.

Srđan Popić, Ištvan Papp and Dejan Đekanović. 2017. Processing cost in case of message parsing on the smart IoT gateway: exploring the costs of unifying the message format to protocol buffer. In: *Proc. of the 2017 International Conference on Smart Systems and Technologies*, 169–173.

Srđan Popić, Dražen Pezer, Bojan Mrazovac and Nikola Teslić. 2016. Performance evaluation of using protocol buffers in the internet of things communication. In: *Proc. of the 2016 International Conference on Smart Systems and Technologies*, 261–265.

Väestörekisterikeskus. 2018. Väestötietojärjestelmän suomalaisten nimiaineistot. https://www.avoindata.fi/data/fi/dataset/none. Luettu 9.12.2018.

Liite 1. Testausohjelman lähdekoodi

Tässä liitteessä esitetään testaustarkoitukseen tuotetun C#-kielisen testiohjelman lähdekoodi. Ajoympäristön tiedot löytyvät luvusta 3. Testiohjelma olettaa projektin lähdekoodien sisältävän koodikatkelmassa 6 esitellystä kuvaustiedostosta käännetyn Person.cs-tiedoston.

Käytetyt NuGet-paketit ja niiden versiot: Google.Protobuf 3.6.1, Google.Protobuf.Tools 3.6.1, MongoDB.Bson 2.7.0 ja Newtonsoft.Json 11.0.2.

Program.cs

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Diagnostics;
 4 using System.Globalization;
 5 using System.IO;
 6 using System.Linq;
 7 using System.Text;
 8 using System.Threading;
 9 using System.Xml.Serialization;
10 using Google.Protobuf;
11 using MongoDB.Bson.IO;
12 using MongoDB.Bson.Serialization;
13 using JsonConvert = Newtonsoft.Json.JsonConvert;
14
15 namespace PerfTest {
16
17     public class Program {
18
19         private static readonly PersonGenerator PersonGenerator =
new PersonGenerator();
20
21         private const string ResultFolder = "results";
22
23         private static TestResult
TestProtobuf(IEnumerable<BasePerson> persons) {
24
25             // Convert to protobuf instances
26             var protoPersons = new List<Person>();
27             foreach (var person in persons) {
28                 protoPersons.Add(person.ToProtoPerson());
29             }
30
31             Console.WriteLine("Protobuf persons ready");
32
33             GC.Collect();
```

```
34
35              var serializedData = new List<byte[]>(1000);
36              var serializedDataSizes = new List<int>(1000);
37
38              // Warmup
39              Console.WriteLine("Warming up & generating data for
deserialization");
40              for (var i = 0; i < 1000; i++) {
41                  foreach (var protoPerson in protoPersons) {
42                      var protoBytes = protoPerson.ToByteArray();
43                      if (serializedData.Count < 1000) {
44                          serializedData.Add(protoBytes);
45
serializedDataSizes.Add(protoBytes.Length);
46                      }
47                  }
48              }
49
50              var serializationResults = new List<TimeSpan>();
51              var deserializationResults = new List<TimeSpan>();
52
53              Console.WriteLine("Serializing");
54              // Serialize
55              for (var i = 0; i < 1000; i++) {
56                  var timer = Stopwatch.StartNew();
57                  foreach (var protoPerson in protoPersons) {
58                      protoPerson.ToByteArray();
59                  }
60                  timer.Stop();
61                  serializationResults.Add(timer.Elapsed);
62              }
63
64              Console.WriteLine("Warming up deserialization");
65              for (var i = 0; i < 1000; i++) {
66                  foreach (var protoBytes in serializedData) {
67                      Person.Parser.ParseFrom(protoBytes);
68                  }
69              }
70
71              Console.WriteLine("Deserializing");
72              // Deserialize
73              for (var i = 0; i < 1000; i++) {
74                  var timer = Stopwatch.StartNew();
75                  foreach (var protoBytes in serializedData) {
76                      Person.Parser.ParseFrom(protoBytes);
77                  }
78                  timer.Stop();
79                  deserializationResults.Add(timer.Elapsed);
80              }
81
```

```
82              return new TestResult(serializationResults,
serializedDataSizes, deserializationResults);
83          }
84
85          private static TestResult TestJson(IEnumerable<BasePerson>
personsEnum) {
86
87              var persons = personsEnum.ToList();
88
89              GC.Collect();
90
91              var serializedData = new List<string>(1000);
92              var serializedDataSizes = new List<int>(1000);
93
94              // Warmup
95              Console.WriteLine("Warming up & generating data for
deserialization");
96              for (var i = 0; i < 1000; i++) {
97                  foreach (var person in persons) {
98                      var serialized =
JsonConvert.SerializeObject(person);
99                      if (serializedData.Count < 1000) {
100                         serializedData.Add(serialized);
101                         // C# uses UTF-16, so the used memory is 2
bytes per character (sizeof(char) == 2)
102                         serializedDataSizes.Add(serialized.Length
* sizeof(char));
103                     }
104                 }
105             }
106
107             var serializationResults = new List<TimeSpan>();
108             var deserializationResults = new List<TimeSpan>();
109
110             Console.WriteLine("Serializing");
111             // Serialize
112             for (var i = 0; i < 1000; i++) {
113                 var timer = Stopwatch.StartNew();
114                 foreach (var person in persons) {
115                     JsonConvert.SerializeObject(person);
116                 }
117                 timer.Stop();
118                 serializationResults.Add(timer.Elapsed);
119             }
120
121             Console.WriteLine("Warming up deserialization");
122             for (var i = 0; i < 1000; i++) {
123                 foreach (var person in serializedData) {
124                     JsonConvert.DeserializeObject(person);
125                 }
```

```
126             }
127 
128             Console.WriteLine("Deserializing");
129             // Deserialize
130             for (var i = 0; i < 1000; i++) {
131                 var timer = Stopwatch.StartNew();
132                 foreach (var person in serializedData) {
133                     JsonConvert.DeserializeObject<BasePerson>(pers
on);
134                 }
135                 timer.Stop();
136                 deserializationResults.Add(timer.Elapsed);
137             }
138 
139             return new TestResult(serializationResults,
serializedDataSizes, deserializationResults);
140         }
141 
142         private static TestResult TestBson(IEnumerable<BasePerson>
personsEnum) {
143 
144             var persons = personsEnum.ToList();
145 
146             GC.Collect();
147 
148             var serializedData = new List<byte[]>(1000);
149             var serializedDataSizes = new List<int>(1000);
150 
151             var memStream = new MemoryStream();
152             var bsonWriter = new BsonBinaryWriter(memStream);
153 
154             // Warmup
155             Console.WriteLine("Warming up & generating data for
deserialization");
156             for (var i = 0; i < 1000; i++) {
157                 foreach (var person in persons) {
158                     BsonSerializer.Serialize(bsonWriter, person);
159                     if (serializedData.Count < 1000) {
160                         serializedData.Add(memStream.ToArray());
161                         serializedDataSizes.Add((int)memStream.Len
gth);
162                     }
163                     // Reset memory stream
164                     memStream.SetLength(0);
165                 }
166             }
167 
168             var serializationResults = new List<TimeSpan>();
169             var deserializationResults = new List<TimeSpan>();
170 
```

```
171             Console.WriteLine("Serializing");
172             // Serialize
173             for (var i = 0; i < 1000; i++) {
174                 var timer = Stopwatch.StartNew();
175                 foreach (var person in persons) {
176                     BsonSerializer.Serialize(bsonWriter, person);
177                 }
178                 timer.Stop();
179                 serializationResults.Add(timer.Elapsed);
180                 // Reset memory stream
181                 memStream.SetLength(0);
182             }
183 
184             bsonWriter.Close();
185             memStream.Close();
186 
187             Console.WriteLine("Warming up deserialization");
188             for (var i = 0; i < 1000; i++) {
189                 foreach (var person in serializedData) {
190                     BsonSerializer.Deserialize<BasePerson>(person)
;
191                 }
192             }
193 
194             Console.WriteLine("Deserializing");
195 
196             // Deserialize
197             for (var i = 0; i < 1000; i++) {
198                 var timer = Stopwatch.StartNew();
199                 foreach (var person in serializedData) {
200                     BsonSerializer.Deserialize<BasePerson>(person)
;
201                 }
202                 timer.Stop();
203                 deserializationResults.Add(timer.Elapsed);
204             }
205 
206             return new TestResult(serializationResults,
serializedDataSizes, deserializationResults);
207         }
208 
209         private static TestResult TestXml(IEnumerable<BasePerson>
personsEnum) {
210 
211             var persons = personsEnum.ToList();
212 
213             GC.Collect();
214 
215             var serializedData = new List<string>(1000);
216             var serializedDataSizes = new List<int>(1000);
```

```
            var serializer = new
XmlSerializer(typeof(BasePerson));
            var memStream = new MemoryStream();

            // Warmup
            Console.WriteLine("Warming up & generating data for
deserialization");
            for (var i = 0; i < 1000; i++) {
                foreach (var person in persons) {
                    serializer.Serialize(memStream, person);
                    if (serializedData.Count < 1000) {
                        serializedData.Add(Encoding.UTF8.GetString
(memStream.ToArray()));
                        serializedDataSizes.Add(memStream.ToArray(
).Length);
                    }
                    memStream.SetLength(0);
                }
            }

            var serializationResults = new List<TimeSpan>();
            var deserializationResults = new List<TimeSpan>();

            Console.WriteLine("Serializing");
            // Serialize
            for (var i = 0; i < 1000; i++) {
                var timer = Stopwatch.StartNew();
                foreach (var person in persons) {
                    serializer.Serialize(memStream, person);
                }
                timer.Stop();
                serializationResults.Add(timer.Elapsed);
                memStream.SetLength(0);
            }

            Console.WriteLine("Warming up deserialization");
            for (var i = 0; i < 1000; i++) {
                foreach (var person in serializedData) {
                    serializer.Deserialize(new
StringReader(person));
                }
            }

            Console.WriteLine("Deserializing");

            // Deserialize
            for (var i = 0; i < 1000; i++) {
                var timer = Stopwatch.StartNew();
                foreach (var person in serializedData) {
```

```
                    serializer.Deserialize(new
StringReader(person));
                }
                timer.Stop();
                deserializationResults.Add(timer.Elapsed);
            }

            return new TestResult(serializationResults,
serializedDataSizes, deserializationResults);
        }

        private static void WriteSizeCsv(List<int> sizes) {
            Console.WriteLine("Writing object sizes");

            using (var writer = new
StreamWriter($@"{ResultFolder}\sizes.csv")) {
                foreach (var size in sizes) {
                    writer.WriteLine($"{size}");
                }
            }
        }

        private static void WriteResultsCsv(string prefix,
TestResult results) {
            Console.WriteLine($"Writing {prefix} test results");

            using (var writer = new
StreamWriter($@"{ResultFolder}\{prefix}_serialization_results.csv")) {
                var timeList =
results.SerializationResults.ToList();
                var sizeList =
results.SerializationSizeResults.ToList();
                for (var i = 0; i < timeList.Count; i++) {
                    // Write microseconds
                    writer.WriteLine($"{((int)(timeList[i].TotalMi
lliseconds *
1000)).ToString(CultureInfo.InvariantCulture)},{sizeList[i]}");
                }
            }

            using (var writer = new
StreamWriter($@"{ResultFolder}\{prefix}_deserialization_results.csv"))
{
                foreach (var result in
results.DeserializationResults) {
                    // Write microseconds
                    writer.WriteLine($"{((int)(result.TotalMillise
conds * 1000)).ToString(CultureInfo.InvariantCulture)}");
                }
```

```
297             }
298         }
299 
300         private static void Main(string[] args) {
301 
302             Console.WriteLine("Generating test data");
303 
304             // Generate 10000 Person instances
305             var persons = new List<BasePerson>();
306             var baseSizes = new List<int>();
307 
308             for (var i = 0; i < 1000; i++) {
309                 var person = PersonGenerator.GeneratePerson();
310                 persons.Add(person);
311                 // Get the (estimated) size of the created person
312                 baseSizes.Add(person.GetRawSize());
313             }
314 
315             Console.WriteLine("Test data generated");
316 
317             // Create result folder
318             if (!Directory.Exists(ResultFolder)) {
319                 Directory.CreateDirectory(ResultFolder);
320             }
321 
322             WriteSizeCsv(baseSizes);
323 
324             Process.GetCurrentProcess().ProcessorAffinity = new
IntPtr(2);
325             Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.High;
326             Thread.CurrentThread.Priority =
ThreadPriority.Highest;
327 
328             Console.WriteLine("-- Protocol Buffers --");
329             var protoResults = TestProtobuf(persons);
330             WriteResultsCsv("proto", protoResults);
331 
332             Console.WriteLine("-- JSON --");
333             var jsonResults = TestJson(persons);
334             WriteResultsCsv("json", jsonResults);
335 
336             Console.WriteLine("-- BSON --");
337             var bsonResults = TestBson(persons);
338             WriteResultsCsv("bson", bsonResults);
339 
340             Console.WriteLine("-- XML --");
341             var xmlResults = TestXml(persons);
342             WriteResultsCsv("xml", xmlResults);
343 
```

```
            Console.WriteLine("Tests complete");
        }
    }
}
```

BasePerson.cs

```
using System.Collections.Generic;

namespace PerfTest {
    public class BasePerson {

        public class City {
            public string Name { get; set; }
            public Location Location { get; set; }
            public string Country { get; set; }

            public City() { }

            public City(string name, Location location, string
country) {
                Name = name;
                Location = location;
                Country = country;
            }

            public int GetRawSize() {
                return Name.Length * sizeof(char) + Country.Length
* sizeof(char) + Location.GetRawSize();
            }
        }

        public class Location {
            public float Latitude { get; set; }
            public float Longitude { get; set; }

            public Location() { }

            public Location(float lat, float lng) {
                Latitude = lat;
                Longitude = lng;
            }

            public int GetRawSize() {
                return 2 * sizeof(float);
            }
        }

        public string FirstName { get; set; }
```

```
41          public string LastName { get; set; }
42          public int Age { get; set; }
43          public City HomeCity { get; set; }
44          public List<string> Interests { get; set; }
45
46          public BasePerson() { }
47
48          public BasePerson(string firstName, string lastName, int
age, City home,
49              List<string> interests) {
50
51              FirstName = firstName;
52              LastName = lastName;
53              Age = age;
54              HomeCity = home;
55              Interests = interests;
56          }
57
58          public int GetRawSize() {
59              var interestsSize = 0;
60              foreach (var interest in Interests) {
61                  interestsSize += interest.Length * sizeof(char);
62              }
63
64              return FirstName.Length * sizeof(char) +
LastName.Length * sizeof(char) + sizeof(int) +
65                      HomeCity.GetRawSize() + interestsSize;
66          }
67
68      }
69  }
```

PersonGenerator.cs

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Text;
 4
 5  namespace PerfTest {
 6      public class PersonGenerator {
 7
 8          private static readonly string[] FirstNames = {
 9              "Timo", "Juha", "Matti", "Kari", "Mikko", "Antti",
"Jari", "Jukka", "Mika", "Markku", "Pekka", "Hannu",
10              "Heikki", "Seppo", "Janne", "Ari", "Sami", "Ville",
"Marko", "Petri", "Lauri", "Jani", "Erkki", "Pentti",
11              "Jorma", "Teemu", "Harri", "Eero", "Raimo", "Jaakko",
"Jarmo", "Risto", "Jussi", "Pasi", "Esa", "Juho",
12              "Pertti", "Martti", "Jouni", "Niko", "Toni", "Arto",
"Reijo", "Veikko", "Markus", "Vesa", "Jouko", "Tomi",
```

```
            "Tuomas", "Tuula", "Anne", "Päivi", "Ritva", "Anna",
"Leena", "Pirjo", "Sari", "Minna", "Marja", "Tiina",
            "Riitta", "Tarja", "Pirkko", "Laura", "Seija", "Eeva",
"Aino", "Liisa", "Raija", "Anja", "Sirpa", "Jaana",
            "Satu", "Heidi", "Hanna", "Sanna", "Eila", "Eija",
"Maria", "Merja", "Johanna", "Kirsi", "Arja", "Maija",
            "Paula", "Ulla", "Sirkka", "Jenni", "Elina", "Katja",
"Mari", "Heli", "Anni", "Nina", "Raili", "Terttu",
            "Hilkka", "Kirsti"
        };

        private static readonly string[] LastNames = {
            "Korhonen", "Virtanen", "Mäkinen", "Nieminen",
"Mäkelä", "Hämäläinen", "Laine", "Heikkinen", "Koskinen",
            "Järvinen", "Lehtonen", "Lehtinen", "Saarinen",
"Salminen", "Heinonen", "Niemi", "Heikkilä", "Kinnunen",
            "Salonen", "Turunen", "Salo", "Laitinen", "Tuominen",
"Rantanen", "Karjalainen", "Jokinen", "Mattila",
            "Savolainen", "Lahtinen", "Ahonen", "Ojala",
"Leppänen", "Väisänen", "Hiltunen", "Kallio", "Miettinen",
            "Leinonen", "Pitkänen", "Aaltonen", "Manninen",
"Koivisto", "Hakala", "Anttila", "Laaksonen", "Hirvonen",
            "Räsänen", "Lehto", "Laakso", "Toivonen"
        };

        private static readonly string[] Interests = {
            "3D-tulostaminen", "Tanssiminen", "Teatteri",
"Ohjelmointi", "Tietokonepelit", "Maalaus", "Origami",
            "Kukkalaitteet", "Jääkiekko", "Jalkapallo",
"Vaeltaminen", "Kalastus", "Autourheilu", "Roolipelaaminen",
            "Infrastruktuuri", "Metsästys", "Shakki", "Nyrkkeily",
"Mahjong", "Golf", "Tennis", "Geokätköily", "Partio",
            "E-urheilu", "Taikuus", "Käsityöt", "Purjehdus",
"Sirkus", "Lukeminen", "Ruoanlaitto", "Ratsastus",
            "Tähtitiede", "Sukututkimus", "Musiikki",
"Mallintaminen", "Huonekasvit", "Liikunta", "Laulaminen"
        };

        private static readonly BasePerson.City[] Cities = {
            new BasePerson.City("Heinola", new
BasePerson.Location(26.0319444444444F, 61.2027777777778F), "Finland"),
            new BasePerson.City("Saarijärvi", new
BasePerson.Location(25.2569444444444F, 62.7055555555556F), "Finland"),
            new BasePerson.City("Posio", new
BasePerson.Location(28.1666666666667F, 66.1111111111111F), "Finland"),
            new BasePerson.City("Miehikkälä", new
BasePerson.Location(27.7F, 60.6708333333333F), "Finland"),
            new BasePerson.City("Mynämäki", new
BasePerson.Location(21.9888888888889F, 60.6791666666667F), "Finland"),
```

```
43              new BasePerson.City("Huittinen", new
BasePerson.Location(22.6986111111111F, 61.1763888888889F), "Finland"),
44              new BasePerson.City("Nousiainen", new
BasePerson.Location(22.0833333333333F, 60.5986111111111F), "Finland"),
45              new BasePerson.City("Pomarkku", new
BasePerson.Location(22.0069444444444F, 61.6930555555556F), "Finland"),
46              new BasePerson.City("Ruovesi", new
BasePerson.Location(24.0680555555556F, 61.9861111111111F), "Finland"),
47              new BasePerson.City("Hämeenlinna", new
BasePerson.Location(24.4652777777778F, 60.9958333333333F), "Finland"),
48              new BasePerson.City("Juuka", new
BasePerson.Location(29.2527777777778F, 63.2416666666667F), "Finland"),
49              new BasePerson.City("Teuva", new
BasePerson.Location(21.7472222222222F, 62.4861111111111F), "Finland"),
50              new BasePerson.City("Iisalmi", new
BasePerson.Location(27.1888888888889F, 63.5611111111111F), "Finland"),
51              new BasePerson.City("Kannus", new
BasePerson.Location(23.9166666666667F, 63.9F), "Finland"),
52              new BasePerson.City("Haapavesi", new
BasePerson.Location(25.3666666666667F, 64.1375F), "Finland"),
53              new BasePerson.City("Sonkajärvi", new
BasePerson.Location(27.5222222222222F, 63.6694444444444F), "Finland"),
54              new BasePerson.City("Vimpeli", new
BasePerson.Location(23.8222222222222F, 63.1611111111111F), "Finland"),
55              new BasePerson.City("Pornainen", new
BasePerson.Location(25.375F, 60.4763888888889F), "Finland"),
56              new BasePerson.City("Veteli", new
BasePerson.Location(23.7888888888889F, 63.4736111111111F), "Finland"),
57              new BasePerson.City("Utsjoki", new
BasePerson.Location(27.0236111111111F, 69.9069444444444F), "Finland"),
58              new BasePerson.City("Laihia", new
BasePerson.Location(22.0111111111111F, 62.9763888888889F), "Finland"),
59              new BasePerson.City("Sodankylä", new
BasePerson.Location(26.5930555555556F, 67.4166666666667F), "Finland"),
60              new BasePerson.City("Kaustinen", new
BasePerson.Location(23.6902777777778F, 63.5486111111111F), "Finland"),
61              new BasePerson.City("Akaa", new
BasePerson.Location(23.8680555555556F, 61.1666666666667F), "Finland"),
62              new BasePerson.City("Taipalsaari", new
BasePerson.Location(28.0597222222222F, 61.1597222222222F), "Finland"),
63              new BasePerson.City("Lumparland", new
BasePerson.Location(20.2583333333333F, 60.1166666666667F), "Finland"),
64              new BasePerson.City("Pyhäranta", new
BasePerson.Location(21.4444444444444F, 60.95F), "Finland"),
65              new BasePerson.City("Alajärvi", new
BasePerson.Location(23.8166666666667F, 63F), "Finland"),
66              new BasePerson.City("Imatra", new
BasePerson.Location(28.7763888888889F, 61.1930555555556F), "Finland"),
67              new BasePerson.City("Jokioinen", new
BasePerson.Location(23.4861111111111F, 60.8041666666667F), "Finland")
```

```
        };

        private readonly Random _rand;

        public PersonGenerator() {
            _rand = new Random(685604657);
        }

        public BasePerson GeneratePerson() {
            // Randomize name
            var firstName =
FirstNames[_rand.Next(FirstNames.Length)];
            var lastName = LastNames[_rand.Next(LastNames.Length)];
            // Randomize age (1-100)
            var age = _rand.Next(1, 101);
            // Randomize city
            var city = Cities[_rand.Next(Cities.Length)];
            // Randomize 1-3 interests
            var interests = new List<string>();
            var interestCount = _rand.Next(1, 4);
            for (var i = 0; i < interestCount; i++) {
interests.Add(Interests[_rand.Next(Interests.Length)]);
            }

            return new BasePerson(firstName, lastName, age, city,
interests);
        }

    }
}
```

ProtobufPersonExtensions.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PerfTest {
    public static class ProtobufPersonExtensions {

        public static Person ToProtoPerson(this BasePerson
basePerson) {
            return new Person {
                FirstName = basePerson.FirstName,
                LastName = basePerson.LastName,
                Age = basePerson.Age,
                City = new City {
                    Name = basePerson.HomeCity.Name,
                    Country = basePerson.HomeCity.Country,
```

```
                    Location = new Location {
                        Latitude =
basePerson.HomeCity.Location.Latitude,
                        Longitude =
basePerson.HomeCity.Location.Longitude
                    }
                },
                Interests = { basePerson.Interests }
            };
        }

    }
}
```

TestResult.cs

```
using System;
using System.Collections.Generic;

namespace PerfTest {
    public class TestResult {

        public IEnumerable<TimeSpan> SerializationResults;
        public IEnumerable<int> SerializationSizeResults;
        public IEnumerable<TimeSpan> DeserializationResults;

        public TestResult(IEnumerable<TimeSpan>
serializationResult, IEnumerable<int> serializationSizeResults,
            IEnumerable<TimeSpan> deserializationResults) {
            SerializationResults = serializationResult;
            SerializationSizeResults = serializationSizeResults;
            DeserializationResults = deserializationResults;
        }

    }
}
```

Pelattavuuden heuristiikat

Juho Penttilä

Tiivistelmä.
Pelattavuus on yleisesti käytetty termi puhuttaessa pelien laadusta. Pelitutkijapiireissä pelattavuuden arviointiin on luotu useita heuristiikkamalleja. Tarkastelen tässä tutkielmassa kolmea erilaista pelattavuuden heuristiseen arviointiin tarkoitettua heuristiikkamallia. Tarkastelen myös mitä mahdollisuuksia pelattavuuden heuristisella arvioinnilla on, mutta myös mitkä asiat voivat olla sen suhteen ongelmallisia.

Avainsanat ja -sanonnat: Pelattavuus, käytettävyys, pelaajakokemus, asiantuntija-arviointi, heuristiikka, playability, gameplay

1. Johdanto

Asiantuntija-arviointi on vakiintunut olennaiseksi osaksi ihmisen ja tietokoneen vuorovaikutuksen tieteenalan työskentelymenetelmiä. Varsinkin Nielsenin [1994] käytettävyysheuristiikat ovat muodostuneet standardiksi arvioitaessa käyttöliittymien toimivuutta. Samaan tapaan, kuin käyttösovellusten käytettävyyteen on kehitetty lukuisia erilaisia asiantuntija-arviointimenetelmiä, on vastaavia ryhdytty kehittämään myös pelisovellusten pelattavuuden arviointiin.

Tähän mennessä esitetyt pelien pelattavuuden asiantuntija-arviointimenetelmät kuitenkin eroavat toisistaan usein jopa huomattavan paljon, mikä johtuu suurelta osin pelien luonteesta varsin monipuolisina ja -ulotteisina sovelluksina. Toisin kuin käyttösovelluksissa, se, miten yksi peli toimii, voi poiketa täysin seuraavasta, joten yhtä kattavia heuristiikkoja kuten esimerkiksi Nielsenin, on todella vaikea, ellei peräti mahdoton luoda. Asiantuntija-arviointimenetelmien erot juontavat sen lisäksi myös arviointimenetelmien kehittäjien erilaisista lähestymistavoista aiheeseen ja jopa itse pelattavuuteen.

Tarkastelen tässä tutkielmassa kolmea erilaista pelattavuuden asiantuntija-arviointimenetelmää, joista jokainen keskittyy pelattavuuden heuristiseen arviointiin. Ensimmäisenä luvussa 2 tarkastelen erilaisia määrittelyjä pelattavuudelle. Seuraavaksi luvussa 3 esittelen tarkasteltavat arviointimenetelmät. Luvun 3 kohdissa tarkastelen jokaista arviointimenetelmää ja niiden erikoispiirteitä tarkemmin. Luvussa 4 pohdin millaisia mahdollisuuksia ja ongelmia käsitellyissä arviointimenetelmissä on ja luvussa 5 kokoan tutkielman havainnot yhteen.

2. Pelattavuuden määrittelyä

Pelattavuus itsessään on merkittävässä roolissa keskusteltaessa lähes mistä tahansa pelaamiseen ja peleihin liittyvästä, ja se on selkeästi eroteltavissa pelien käytettävyyden käsitteestä. Esimerkiksi Febretti ja Garzotto [2009] löysivät korrelaatiota nimenomaan hyvän pelattavuuden myönteisiin vaikutuksiin pitkäaikaisissa tietokonepelikokemuksissa, kun taas samassa kontekstissa pelin käytettävyys korreloi vain hyvin vähän.

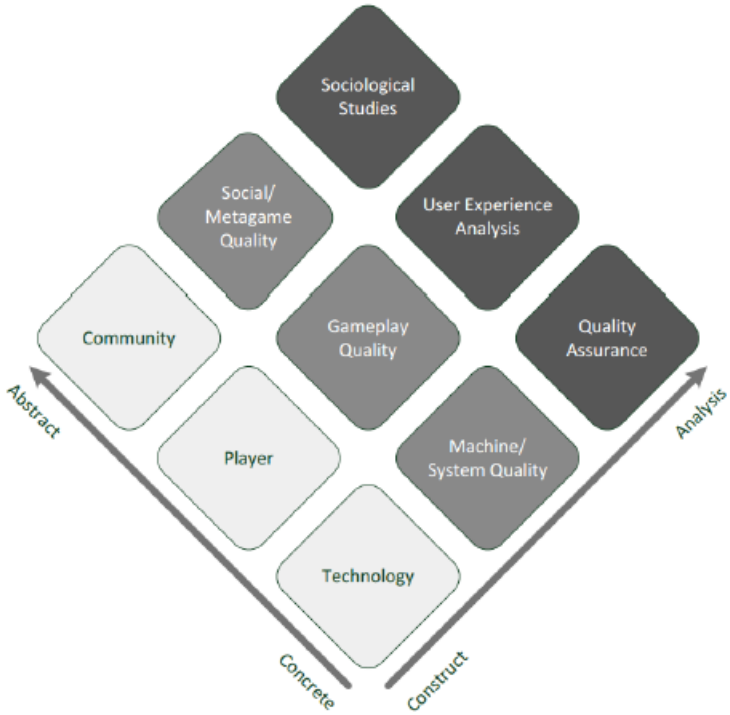
Siinä, missä käytettävyys on jopa suhteellisen yksioikoinen ja selkeästi määritelty termi, on pelattavuus monitulkintainen ja selvästi vaikeammin määriteltävissä. Koska jokaisella tässä tutkielmassa käsiteltävällä asiantuntija-arviointimenetelmien kehittäjällä ja jopa tutkielman lukijalla saattaa olla vähän erilainen käsitys pelattavuudesta, mikä luonnollisesti vaikuttaa myös heidän lähestymistapaansa sen arvioinnissa, on tarpeen ensin tarkastella muutamaa esimerkkiä siitä, mitä pelattavuudella voidaan tarkoittaa.

Pelattavuudesta terminä tekee tavallista monimutkaisemman suomenkielisessä kontekstissa se, että kyse on englanniksi kahdesta eri termistä. Englanninkieliset termit playability sekä gameplay molemmat kääntyvät suomeksi vain pelattavuudeksi, mutta niillä on eri merkitykset. Gameplay tarkoittaa sitä, kuinka sujuvaa ja toimivaa itse pelaaminen on, esimerkiksi kuinka hyvin kontrollit tottelevat pelaajaa, ja on usein käytetty termi englanninkielisissä peliarvosteluissa. Playability taas on termi, joka usein sisältää myös gameplayn käsitteen, mutta sen lisäksi tarkoittaa kaikkea sen ympärillä, mikä vaikuttaa siihen, kuinka hyvä tai luonteva peliä on pelata. Keskityn tässä tutkielmassa tarkastelemaan pelattavuutta lähinnä playability-käsitteen puitteissa, mutta myös gameplay-puoli nousee aika ajoin väistämättä esiin, sillä ne risteävät keskenään myös englanninkielisissä tieteellisissä teksteissä.

Pelattavuus on terminä niin laajassa käytössä tieteellisen yhteisön ulkopuolella, että on hyvä pohtia ensiksi sen maallikkomaisempia käyttötapoja. Pelattavuus esiintyy usein muun muassa pelijournalistisissa teksteissä, mutta sen käyttötapa saattaa vaihdella suurestikin kontekstista riippuen. Usein puhutaan hyvästä ja huonosta pelattavuudesta, mutta mikä lopulta on hyvää ja huonoa pelattavuutta, voi riippua täysin muun muassa kirjoittajan omista mieltymyksistä, mutta myös tarkasteltavan pelin genrestä. Jos esimerkiksi pelihahmo toimintapelissä tottelee pelaajan kontrolleja tarkasti ja pelaaja tuntee olevansa tilanteiden hallinnassa, puhutaan usein hyvästä pelattavuudesta, mutta vastavuoroisesti vuoropohjaisessa strategiapelissä niillä asioilla ei ole niin paljoa merkitystä. Siinä tapauksessa hyvästä pelattavuudesta saatetaan puhua esimerkiksi, jos tarvittavat komennot ovat tarpeeksi helposti löydettävissä ja

mahdollisuudet toteuttaa erilaisia strategioita ovat riittävän laajat. Jälkimmäisessä esimerkissä voi myös huomata, kuinka hyvä pelattavuus saattaa silloin tällöin muistuttaa myös hyvän käytettävyyden määrittelyä.

Vaikka pelattavuus on yleinen termi niin peliharrastajien kielessä, pelijournalistissa lähteissä kuin pelitutkijoidenkin termistössä, eivät edes pelitutkijat ole onnistuneet luomaan sille yhtä oikeaa määritelmää [Paavilainen 2017]. Tämä on erikoista, sillä käsitteenä pelattavuus on ollut olemassa tieteellisenä konseptina jo pitkään. Yksi ensimmäisistä maininnoista löytyy Leitmannin [1974] nollasummapelejä käsittelevästä artikkelista. Tosin, vaikka sana ilmenee artikkelissa jo nykyään tunnetussa playabilityn muodossa, se tarkoittaa siinä enemmän kirjaimellista pelattavuutta, esimerkiksi onko jokin tietty liike pelattavissa jossain tietyssä tilanteessa, eikä niinkään viittaa pelin tai pelijärjestelmän laatuun millään tapaa.



Kuva 1. Hierarkkinen pelien käytettävyyden malli. [Nacke 2009]

Nacke [2009] lähestyy aihetta hieman poikkeavasta näkökulmasta ja pelattavuuden tilalle esittää hierarkkisen mallin pelien käytettävyydelle (kuva 1). Hän argumentoi ratkaisuaan sillä, että suurin osa tiedeyhteisön määrittelemistä pelattavuuden malleista rakentuu muutenkin suoraan käytettävyyden konseptin päälle. Nacken hierarkkisessa mallissa teknologia on jokaisen digitaalisen pelin perusta. Vasemmalla akselilla on kuvattuna mitattavat entiteetit konkreettisesta abstraktiin, oikealla akselilla niiden kuvaajat teoreettisesta konstruktiosta käytännölliseen soveltamiseen. Valitettavasti Nacke ei perehdytä lukijaa

kirjoituksessaan juurikaan sen syvemmälle mallinsa taustoihin tai mahdollisiin käyttötapoihin, vaan ainoastaan tyytyy mainitsemaan jatkokehityksen tarpeesta.

Paavilainen [2017] puolestaan pyrkii määrittelemään pelattavuuden pelikeskeisenä terminä. Sen tärkeimpänä ominaisuutena on, että pelattavuus terminä keskittyisi vain pelin sisäisiin ominaisuuksiin, eikä ulottuisi esimerkiksi sosiaalisiin konteksteihin tai vaikkapa pelaajakokemuksiin. Hänen mukaansa pelattavuuden tyylisen kvalitatiivisen termin tulisi keskittyä peliin itseensä, eli sellaisiin asioihin, mitä pelisuunnittelija pystyy peliinsä tarkoituksellisesti luomaan.

Paavilaisen pelikeskeisessä mallissa on kolme pääosaa: funktionaalisuus, joka mittaa pelin teknistä laatua, kuten lataustaukojen pituutta, ohjelmointivirheiden määrää tai esimerkiksi tasaista ruudunpäivitystä, käytettävyys, joka keskittyy pelin käyttöliittymän toimivuuteen ja intuitiivisuuteen, sekä pelattavuus, joka tässä tapauksessa tarkoittaa gameplay-termin määrittelemää pelattavuutta, eli sitä, kuinka loogisesti ja tasapainoisesti pelin säännöt ja mekaniikat toimivat. Pelikeskeinen pelattavuuden malli pyrkii siis täysin objektiiviseen pelattavuuden tarkastelemiseen jättäen henkilön subjektiiviset kokemukset mallin ulkopuolelle, mikä olisi myös erinomainen ja selkeä lähtökohta heuristiikkojen määrittelyyn.

3. Heuristiikkamalleja

Tämän luvun kohdissa käsittelen kolmea erilaista pelattavuuden heuristiikkakokoelmaa. Ensiksi tarkastelen pelattavuuden heuristisen arvioinnin mallia [Desurvire *et al.* 2004]. Sen jälkeen on vuorossa Korhosen ja Koiviston [2006] pelattavuuden heuristiikat mobiilipeleille. Lopuksi tarkastelen pelattavuusheuristiikkojen luomista perustuen internetin peliarvostelujen substantiiveihin ja adjektiiveihin [Zhu *et al.* 2017].

3.1. Pelattavuuden heuristinen arviointi

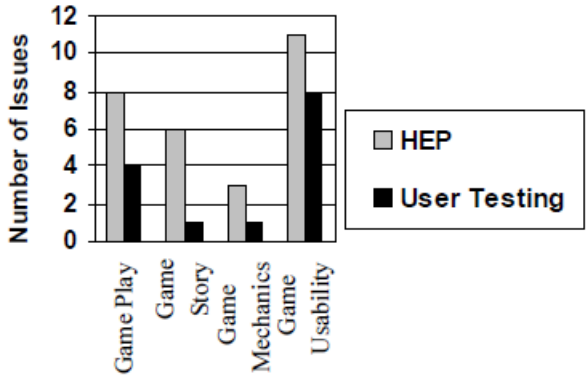
Ensimmäiset pelattavuuden heuristiset arviointimallit syntyivät jo ennen vuosituhannen vaihdetta. Esimerkiksi Malone [1982] kehitti omat heuristiikkansa opetuspeleihin jo 80-luvulla.

Aiemmista yrityksistä huolimatta pelattavuuden heuristiikkoja ei kuitenkaan oltu vielä koottu näin kattavaksi ja kokonaisvaltaiseksi heuristiikkojen listaksi ennen kuin Desurvire ja muut [2004] kokosivat Heuristic Evaluation for Playability -heuristiikkakokoelman (liite 1), joka lyhentyi kätevästi muotoon

HEP. HEP perustuu tuotanto- ja pelitestausheuristiikkojen kirjallisuuteen, ja on tekijöidensä mukaan soveltuva arvioimaan sekä video- että lautapelejä.

HEP on jaettu neljään pääosaan, joista jokaisessa on seitsemästä kuuteentoista yksittäistä heuristiikkaa. Game Play sisältää kuusitoista heuristiikkaa, jotka keskittyvät pelaajan pelikokemuksiin. Game Storyn kahdeksan heuristiikkaa käsittelee pelin tarinallisia elementtejä ja sitä, saako peli pelaajan kiinnostumaan tarinastaan. Mechanicsin seitsemän heuristiikkaa keskittyy pelin mekaanisiin elementteihin, kuten esimerkiksi toimiiko pelin tekoälyhahmot pelaajan odottamalla tavalla tai että onko pelin tavoitteet tai tilanne tarpeeksi hyvin näkyvillä pelaajalle. Viimeiseksi Usability sisältää kaksitoista pelin käytettävyyteen liittyvää heuristiikkaa.

Testatakseen HEP:n toimivuutta, Desurvire ja muut [2004] käyttivät heuristiikkojaan uuden pelin arviointiin sen kehitysvaiheessa, ja vertasivat tuloksiaan olemassa olleisiin pelitestauksen käyttäjätestausmetodeihin. Testattava peli oli prototyyppivaiheessa oleva demo, jossa pystyi siirtymään näkymästä toiseen ympäri peliä, mutta varsinaista pelattavaa ei oltu vielä implementoitu.



Kaavio 1. HEP-mallin ja käyttäjätestauksen löytämät pelattavuusongelmat. [Desurvire *et al.* 2004]

Testauksen lopputuloksena sekä HEP että käyttäjätestaus löysivät paljon samoja ongelmia, mutta lukumääräisesti HEP paljasti niitä enemmän jokaisessa kategoriassa (kaavio 1). Suhteessa heuristiikkojen lukumäärään HEP-malli onnistui paikantamaan paljon ongelmia etenkin tarinan ja käytettävyyden osa-alueissa. Mekaniikkojen ja Game Playn puolesta osumatarkkuus oli pienempi, mikä selittyy pitkälti sillä, että peli oli vielä niin varhaisessa vaiheessa, että siinä oli paljon mekaaniseen ja pelaajakokemukseen liittyviä asioita, mitä ei päässyt vielä arvioimaan. Näiden tulosten perusteella Desurvire ja muut arvioivat, että

HEP olisi tehokas työkalu etenkin alkupään kehityksen aikana pelin tarinan ja käytettävyyden arvioinnissa.

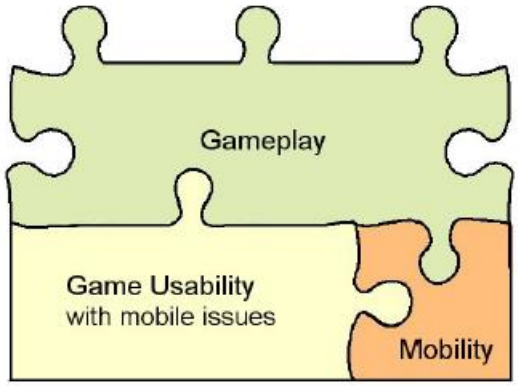
Verrattuna käyttäjätestaukseen HEP-mallin avulla havaittiin useita sellaisia ongelmia, mitä käyttäjätestissä ei lainkaan, mutta oli myös joitain sellaisia, jotka havaittiin ainoastaan seuraamalla oikeiden pelaajien pelaamista käyttäjätestitilanteessa. Käyttäjätestit onnistuivat myös havainnoimaan asioita, joita heuristisella arvioinnilla ei voi seurata, kuten sitä, onko pelaaja tylsistynyt, kuinka hyvin pelin haaste vastasi pelaajan taitoja, kuinka pelin rytmitys toimi pelaajalle ja kuinka hyvin pelaaja ymmärsi pelin terminologiaa.

Desurvire ja muut [2004] toteavat, että HEP osoittaa olevansa tehokas työväline pelisuunnittelun alkuvaiheilla ja sillä pystyy löytämään tarkasti hyvin spesifejä ongelmia jo varhaisessa vaiheessa. Siitä huolimatta HEP ei voi täysin korvata pelitestausta, sillä oikean ihmisen käyttäytyminen on arvaamatonta eikä sitä pysty täydellisesti ennustamaan ennalta.

3.2. Pelattavuuden heuristiikat mobiilipeleille

Kaikki pelattavuusheuristiikkakokoelmat eivät edes pyri olemaan lähellekään niin kaikenkattavia, kuin esimerkiksi HEP. Koska pelit ovat monimuotoinen alue jo itsessään, saattaa olla aivan perusteltua suunnitella erilaisia heuristiikkakokoelmia erilaisiin käyttötarkoituksiin. Juuri siksi Korhonen ja Koivisto [2006] pyrkivät luomaan oman heuristiikkakokoelmansa tarkennettuna mobiilipelien pelattavuuden arviointiin, sillä etenkin mobiilipelit toimivat usein hyvin erilaisilla säännöillä ja erilaisessa kontekstissa muihin peleihin verrattuna.

Nykysilmin tarkasteltuna kannattaa huomioida, että nämä mobiilipeliheuristiikat on luotu aikana, jolloin mobiililaitteet olivat vielä kovin erilaisia, eivätkä esimerkiksi kosketusruudut olleet lähellekään yhtä suosittuja kuin nykyään. Sen vuoksi Korhonen ja Koivisto [2006] olettavat, että vuorovaikutus pelin kanssa tapahtuu aina perinteisillä numeronäppäimillä ja muutamalla mahdollisella laitemallista riippuvalla valintapainikkeella, tai joskus harvoin pienellä sauvaohjaimella. Myös mobiililaitteen ruudun koko ja sitä myöten pelinäkymä olivat tuolloin vielä huomattavasti pienempiä kuin nykyään.



Kuva 2. Mobiilipelien arviointimalli. [Korhonen and Koivisto 2006]

Korhosen ja Koiviston mobiilipelien heuristiikkakokoelma perustuu kolmiosaiseen modulaariseen arviointimalliin (kuva 2). Mallin osat ovat gameplay, joka käsittää muun muassa pelin mekaaniset osuudet, game usability, jota voi käyttää pelin käytettävyyden arviointiin, ja mobility, joka keskittyy käsittelemään mobiilipelien liikuteltavan luonteen aiheuttamia ongelmia. Yhtenä maininnan arvoisena erona HEP-malliin on se, että HEP:ssä pelin tarinaa käsitellyt osuus on Korhosen ja Koiviston mallissa yhdistetty gameplayn sisälle. Korhosen ja Koiviston mallin modulaarinen luonne tarkoittaa sitä, että jokainen osio on luotu omaksi erilliseksi kokonaisuudekseen, jota voi myös käyttää erikseen vain jonkin tietyn alueen arviointiin.

Aluksi Korhonen ja Koivisto kehittivät tiiviimmän yhdentoista heuristiikan listan. He halusivat pitää listan tiiviinä, sillä lyhyt lista on helpompi muistaa ja sitä on helpompi hyödyntää asiantuntija-arvioinnin apuvälineenä. He alkoivat testata heuristiikkakokoelmaansa tuotantovaiheessa olevalla älypuhelinpelillä, mutta huomasivat nopeasti, että tiivis kokoelma ei ole lainkaan riittävä peliarviointeihin. Alustavan kokoelman avulla he onnistuivat paikantamaan 63 ongelmaa, joista 16 eivät sopineet mihinkään alkuperäisen kokoelman heuristiikoista, vaikka ne oli tarkoituksella luotu mahdollisimman yleispäteviksi. Ensimmäisten testien havaintoihin perustuen Korhonen ja Koivisto loivat 18 uutta heuristiikkaa ja lisäsivät ne vanhan kokoelman jatkoksi. Lopullisen heuristiikkakokoelman kokonaismääräksi tuli näin ollen 29 kappaletta. Kolmeen osaan jaetut heuristiikat vastaavat aiemmin mainittua modulaarista mallia, koostuen game usabilitystä (taulukko 1), mobilitystä (taulukko 2) ja gameplaystä (taulukko 3).

No.	Game Usability Heuristics
GU1	Audio-visual representation supports the game
GU2	Screen layout is efficient and visually pleasing
GU3	Device UI and game UI are used for their own purposes
GU4	Indicators are visible
GU5	The player understands the terminology
GU6	Navigation is consistent, logical, and minimalist
GU7	Control keys are consistent and follow standard conventions
GU8	Game controls are convenient and flexible
GU9	The game gives feedback on the player's actions
GU10	The player cannot make irreversible errors
GU11	The player does not have to memorize things unnecessarily
GU12	The game contains help

Taulukko 1. Käytettävyyden heuristiikat. [Korhonen and Koivisto 2006]

No.	Mobility Heuristics
MO1	The game and play sessions can be started quickly
MO2	The game acommodates with the surroundings
MO3	Interruptions are handled reasonably

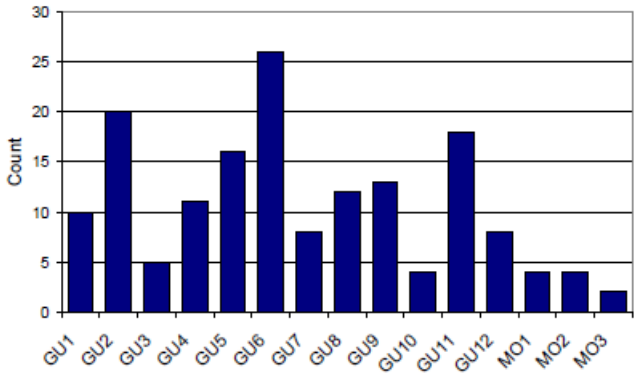
Taulukko 2. Liikuteltavuuden heuristiikat. [Korhonen and Koivisto 2006]

No.	Gameplay Heuristics
GP1	The game provides clear goals and supports player-created goals
GP2	He player sees the progress in the game and can compare the results
GP3	The players are rewarded and rewards are meaningful
GP4	The player is in control
GP5	Challenge, strategy, and pace are in balance
GP6	The first-time experience is encouraging
GP7	The game story supports the gameplay and is meaningful
GP8	There are no repetitive or boring tasks
GP9	The players can express themselves
GP10	The game supports different playing styles
GP11	The game does not stagnate
GP12	The game is consistent

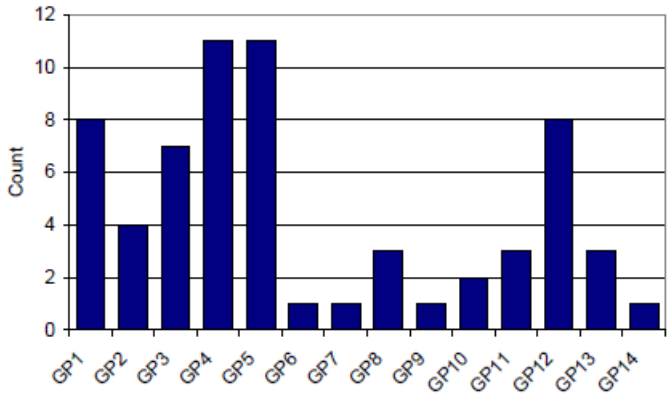
GP13	The game uses orthogonal unit differentiation
GP14	The player does not lose any hard-worn possessions

Taulukko 3. Pelimekaaniset heuristiikat [Korhonen and Koivisto 2006]

Vahvistaakseen laajennetun heuristiikkakokoelman toimivuuden, Korhonen ja Koivisto [2006] käyttivät sitä viiden erilaisen tuotantovaiheessa olevan pelin heuristiseen arviointiin. Yhteensä kaikista viidestä pelistä he onnistuivat paikantamaan tämän heuristiikkakokoelman avulla 235 mahdollista pelattavuusongelmaa, joista 151 liittyi pelien käytettävyyteen ja 10 liikuteltavuuteen (kaavio 2). 64 havaituista ongelmista liittyi pelien mekaniikkoihin (kaavio 3).



Kaavio 2. Viiden testipelin havaitut käytettävyys- ja liikuteltavuusongelmat. [Korhonen and Koivisto 2006]



Kaavio 3. Viiden testipelin havaitut pelimekaaniset ongelmat. [Korhonen and Koivisto 2006]

3.3. Pelattavuusheuristiikat perustuen internetin peliarvosteluiden substantiiveihin ja adjektiiveihin.

Desurviren ja muiden [2004] sekä Korhosen ja Koiviston [2006] lisäksi lukuisat muutkin pelitutkijat ovat pyrkineet luomaan omia peliheuristiikkakokoelmiaan perinteisemmillä suunnittelumenetelmillä [esim. Malone 1982, Pinelle *et al. 2008*]. Se ei kuitenkaan ole ainut mahdollinen lähestymistapa heuristiikkojen määrittelyyn. Zhu ja muut [2017] pyrkivät hyödyntämään sanastollisen analyysin keinoja luodakseen kattavamman ja luotettavamman heuristiikkakokoelman kuin heitä edeltäneet tutkijat.

Zhu ja muut argumentoivat, että aiemmat peliheuristiikkakokoelmat sisältävät kolme suurta vikaa. Heidän mukaansa ehdotetut heuristiikkakokoelmat perustuvat ensinnäkin liian pieneen datajoukkoon minkä vuoksi niistä voi helposti jäädä puuttumaan kriittisiäkin ongelmia. Toiseksi kokoelmat perustuvat kvalitatiiviseen tietoon, eivätkä empiirisesti vahvistettuun aineistoon. Kolmanneksi heidän mukaansa heuristiikkakokoelmat keskittyvät usein turhan keskitettyyn otantaan peleistä, eikä niitä silloin voi hyödyntää yleisemmässä mittakaavassa.

Zhu ja muut ehdottavat omaa heuristiikkakokoelmaansa, jonka kasaamiseen on hyödynnetty sanastollista analyysiä. Heidän keräämä heuristiikkakokoelmansa perustuu aiempaan tutkimukseen, jossa he keräsivät internetin videopeliarvosteluista adjektiiveja ja pyrkivät niiden perusteella kokoamaan vastaavan kokoelman [Zhu *et al.* 2013, Zhu and Fang 2014]. Vaikka pelkkien adjektiivien sanastollinen analyysi paljastikin pelityylien yleisiä piirteitä, se ei kuitenkaan riittänyt paljastamaan niiden kontekstia, joten Zhu ja muut [2017] lisäävät uudessa mallissa analyysiin mukaan substantiivit. He argumentoivat, että adjektiivien ja substantiivien yhdistetty sanastollinen analyysi hyödyttää kahdella tavalla: se paitsi paljastaa internetarvioiden keskinäisiä kaavoja, mutta myös kontekstirikasta tietoa liittyen pelaajakokemuksen erilaisiin aspekteihin.

Tutkimuksessaan Zhu ja muut [2017] analysoivat sanastollista analyysiä hyödyntäen 821122:ta peliarvostelua IGN, Gamestop ja Gamespot -sivustoilta. Arvosteluaineistosta he saivat eristettyä 21535 erillistä substantiivia ja 4327 elijargoniin liittyvää termiä. Niistä karsittiin harvinaisimmat ja vähiten käytetyt pois niin, että jäljelle jäi 4342 termiä, jotka yhdistettiin aiemmassa tutkimuksessa [Zhu *et al.* 2013] kerättyyn adjektiivilistaan. Siitä joukosta saatiin eristettyä 97 faktoria faktorianalyysin avulla. Näistä faktoreista tunnistettiin potentiaaliset heuristiikat, luotiin alustava heuristiikkalista ja jalostettiin lopullinen pelattavuusheuristiikkakokoelma, kooltaan 90 kappaletta (liite 2).

Aikaisempiin peliheuristiikkatutkimuksiin verrattuna Zhun ja muiden [2017] heuristiikkakokoelma sisältää 37 aiemmin havaittua, mutta myös 53 täysin uutta heuristiikkaa. He argumentoivat, että 90:n heuristiikan löytyminen osoittaa sanastollisen analyysin olevan tehokas lähestymiskeino aiheeseen. Heidän mukaansa analyysin avulla löydetyt sanastomallit tarjoavat hyvin spesifiä kontekstuaalista tietoa, mitä voi käyttää heuristiikkojen luomiseen. Zhu ja muut [2017] argumentoivat heidän luomiensa heuristiikkojen sisältävän viisi päävahvuutta: ne perustuvat valmiiksi todella suureen aineistoon, faktorianalyysin ansiosta ne perustuvat kaikkein kriittisimpiin potentiaalisiin ongelmiin, ne tarjoavat paljon spesifiä tietoa, ne ovat kattavia ja ne perustuvat pitkälti pelaajien itsensä käyttämään kieleen.

4. Heuristiikkamallien mahdollisuuksia ja ongelmia

Aiemmassa luvussa havaittiin, kuinka erilaisia lähestymistapoja pelattavuuden heuristiikkakokoelmien luomiseen on nähty muun muassa tutkijapiireissä. Olen tyytyväinen, että esiteltyjen heuristiikkakokoelmien tekijät ovat myös nähneet vaivaa osoittaakseen heuristiikkojensa toimivuutta oikeiden pelien pelattavuutta arvioidessa tai perustaneet heuristiikkansa niin suureen määrään valmista tietoa, että se tukee heidän argumenttejaan.

Heuristiikat voivat olla parhaimmillaan todella vahva työkalu pelien pelattavuuden arvioinnissa. Niitä voi käyttää paitsi pelattavuusongelmien nopeaan ja tehokkaaseen havainnointiin [Korhonen and Koivisto 2006], mutta ne toimivat myös suuntaa-antavina ohjenuorina toimivan pelisuunnittelun edistämiseksi jo ennen kalliiden prototyyppien valmistusta [Desurvire *et al.* 2004].

Pelattavuusheuristiikat ovat siis hyödyllisiä, mutta etenkin yksi yksityiskohta kiinnitti huomioni tutkiessani erilaisia heuristiikkamalleja: todella usein heuristiikkojen kehittäjät painottivat, että heuristiikat yksinään eivät voi korvata käyttäjätestausta. Esimerkiksi Desurvire ja muut [2004] mainitsevat, että ainoastaan käyttäjätestauksella pystyttiin havaitsemaan joitain tiettyjä käyttäjäkohtaisia ilmiöitä. Heidän luoma HEP-mallinsa onnistui kyllä löytämään enemmän mahdollisia pelattavuusongelmia jokaisessa eritellyssä kategoriassa, mutta jos malli ei voi ottaa huomioon ihmisen arvaamatonta käytöstä tai asioita, joita voi todeta vain pelitestauksella, eivät heuristiikat itsekseen voi koskaan täysin korvata pelitestausta, vaan ainoastaan toimia sen tukena ja tehokkaina ohjesääntöinä pelisuunnittelun alkuvaiheilla.

Korhonen ja muut [2009] kertovat muista pelattavuusheuristiikkojen mahdollisista ongelmista. He vertasivat sekä HEP-mallia [Desurvire *et al.* 2004] että Korhosen ja Koiviston [2006] mobiilipelien heuristiikkamallia keskenään, ko-

koamalla kahdeksanhenkisen ryhmän pelisuunnittelijoista ja -tutkijoista, joka jaettiin kahteen erilliseen pienryhmään ja heitä ohjeistettiin käyttämään toista heuristiikkakokoelmista testikappaleena toimineen mobiilipelin pelattavuuden asiantuntija-arviointiin. Testin tuloksena osanottajat sanoivat olevansa tyytyväisiä heuristisen arvioinnin tehokkuuteen ja mahdollisuuksiin pelattavuusongelmien havaitsemiseen jo varhaisissa suunnitteluvaiheissa, mutta huomioivat, että heuristiikkojen käyttäminen pelien kontekstissa ei välttämättä ole niin yksinkertaista, kuin käyttösovelluksissa johtuen oikean abstraktiotason määrittelemisen vaikeudesta.

Erilaisten pelien ja pelisuunnittelutyylien kirjo on niin laaja, että oikean abstraktiotason löytäminen heuristiikkojen luomisessa on välttämätöntä: liian yleispätevät heuristiikat eivät tarjoa tarpeeksi informaatiota ongelmien löytämiseen, kun taas liian tarkat heuristiikat eivät välttämättä sovellu kovinkaan monen eri tyylin pelin heuristiseen arviointiin. Hyvänä esimerkkinä HEP-mallista [Desurvire *et al.* 2004] kokonainen yksi osio (liite 1) on omistettu täysin pelin tarinankerronnan heuristiikoille, vaikka on aivan täysin normaalia, että monissa pelityyleissä, kuten urheilu- tai ongelmanratkontapeleissä, ei ole lainkaan tarinallisia elementtejä, eikä kyseisistä heuristiikoista olisi siten mitään hyötyä kyseisten pelityylien pelejä arvioitaessa.

Oikean abstraktiotason löytäminen on selvästi suurin haaste pelattavuusheuristiikkojen kokoamisessa. Etenkin vuosituhannen alun heuristiikkamalleissa näkyy selvästi, että niiden tekijät ovat hakeneet innoitusta Nielsenin [1994] yksinkertaisen tehokkaasta ja tiivistetystä, mutta silti äärimmäisen toimivasta käytettävyysheuristiikkalistasta. Muun muassa Korhonen ja Koivisto [2006] pyrkivät ensimmäisellä yrityksellään tiiviiseen yhdentoista heuristiikan kokoelmaan, mutta havaitsivat itsekin nopeasti, että se on aivan liian suppea ollakseen tehokas työkalu, ja päätyivät kasvattamaan sen 29:ään (taulukot 1, 2 ja 3).

Abstraktiotason toisessa äärilaidassa on Zhun ja muiden [2017] kokoama järkälemäinen 90:n heuristiikan kokoelma (liite 2). Sen etuna on, että se perustuu niin suureen aineistoon, että jokainen kokoelman heuristiikka on todistetusti toimiva sääntö. Sen suurin heikkous taas on, että luomistavasta juontuva abstraktiotaso on niin yksityiskohtainen, että monet heuristiikat eivät toimi kovinkaan moneen peliin. Esimerkiksi heuristiikat PB#37 ja PB#49 keskittyvät vain tiettyjen peligenrejen hyvin spesifeihin ongelmiin.

Lisäongelman pelattavuusheuristiikkojen luomiselle tuo mukanaan se, että peliala elää jatkuvassa muutoksessa: pelien ja pelilaitteiden teknologia kehittyy ja pelattavuuteen vaikuttavat standardit muun muassa ohjaustapojen kanssa

niiden mukana. Hyvä esimerkki on Korhosen ja Koiviston [2006] mobiilipelien pelattavuuden heuristiikat, jotka oli luotu vanhojen mobiililaitteiden ehdoilla: ruudut olivat pieniä ja jokaisessa laitteessa pystyi olettaa olevan numeronäppäimet ja useampi erillinen valintapainike. Alkuperäinen iPhone kuitenkin julkaistiin jo heti seuraavana vuonna, ja mullisti ilmestyttyään kaiken sen, miten matkapuhelimia käytettiin: painikkeet katosivat ja korvautuivat entistä suuremmilla kosketusnäytöillä. Siinä, missä Nielsenin [1994] käytettävyysheuristiikkoja on voitu hyödyntää muuttumattomina jo yli 20 vuotta, ei pelattavuusheuristiikkojen kestosta ole toistaiseksi mitään varmuutta. Myös Zhun ja muiden [2017] kokoamassa heuristiikkakokoelmassa on huomattavissa kerätyn aineiston aikakauden vaikutus itse heuristiikkoihin. Esimerkiksi heuristiikka PB#31 (liite 2) mainitsee ohjaimessa sijaitsevien ruutujen positiivisen vaikutuksen tietyssä peligenressä, mutta kyseessä ei ole pelialan standardi, vaan ohjainspesifit näytöt ovat Nintendon Wii U -konsolin erikoisuus. Wii U itsessään on jo korvautunut Nintendon seuraavan konesukupolven laitteella ja oli aktiivisimmillaan vuodesta 2012 vuoteen 2016 asti.

Pelattavuus ei siis ole kovin helppo aihe heuristiseen arviointiin: se on laaja, vaihteleva ja alati kehittyvä käsite, jonka arvioinnin tiivistäminen heuristiikkojen tapaisiin nyrkkisääntöihin on äärimmäisen vaikeaa. Tuoreemmat yritykset, kuten Zhun ja muiden [2017] sanastollisella analyysillä kootut heuristiikkakokoelmat tarjoavat uusia näkökulmia entistä parempien heuristiikkamallien luontiin, mutta ne kaipaavat vielä lisäkehitystä.

Uskon, että sanastollinen analyysi on kuitenkin hyvä tapa lähteä kartoittamaan mahdollisia pelattavuusheuristiikkoja. Sen lisäksi, että sillä löytyvät heuristiikat ovat usein lähes pelikohtaisia, on myös aivan mahdollista, että joitain tärkeitä potentiaalisia pelattavuusongelmia saattaisi jäädä havaitsematta aineiston suuresta koosta huolimatta. Siksi yksi peliheuristiikkatutkimuksen seuraavista askelista saattaisi olla lähteä kartoittamaan, voiko sanastollisella analyysillä luodun alustavan heuristiikkakokoelman pohjalta kasata perinteisemmillä tavoilla lopullista ja toimivampaa heuristiikkojen mallia.

5. Yhteenveto

Tämän tutkielman aikana olen tarkastellut kolmea erilaista pelattavuuden heuristisen arvioinnin mallia. Olen luonut yleiskatsauksen niiden erikoisuuksiin ja siihen, miten kunkin kehittäjä on pyrkinyt vahvistamaan heuristiikkojensa paikkansapitävyyden.

Olen myös käsitellyt pelattavuuden heuristiikkojen mahdollisuuksia ja ongelmia. Olen todennut heuristiikkojen olevan hyvä työväline pelattavuuden

ongelmien havaitsemiseen jo pelien varhaisissa kehitysvaiheissa. Pelattavuusheuristiikat ovat myös vahva tukiväline perinteisen käyttäjätestauksen oheen, mutta ne eivät voi ikinä kokonaan korvata käyttäjätestausta, sillä heuristiikoilla ei voi todeta kaikkea sitä, mitä seuraamalla oikean ihmisen toimia voi.

Suurimpina ongelmina pelattavuuden heuristiikoilla on oikein abstraktiotason valitseminen sekä pelien alati kehittyvät standardit niin tekniikan kuin pelikehityksenkin puolella. Abstraktiotaso on etenkin todella tärkeä, sillä liian yleismaalliset heuristiikat eivät tarjoa tarpeeksi tietoa, kun taas liian spesifit heuristiikat ovat käyttökelpoisia vain hyvin pieneen osaan peleistä.

Viiteluettelo

Heather Desurvire, Martin Caplan and Jozsef A. Toth. 2004. Using heuristics to evaluate the playability of games. In: *Proc. of the CHI '04. Extended Abstracts on Human Factors in Computing Systems (CHI EA '04)*, 1509–1512.

Alessandro Febretti and Franca Garzotto. 2009. Usability, playability and long-term engagement in computer games. In: *Proc. Of the CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*, 4063–4068.

Hannu Korhonen and Elina M. I. Koivisto. 2006. Playability heuristics for mobile games. In: *Proc. of the 8th Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '06)*, 9–16.

Hannu Korhonen, Janne Paavilainen and Hannamari Saarenpää. 2009. Expert review method in game evaluations: comparison of two playability heuristic sets. In: *Proc. of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era (MindTrek '09)*, 74–81.

George Leitmann. 1974. On some consequences of playability in differential games. In: *Proc. of the IEEE Conference on Decision and Control Including the 13th Symposium on Adaptive Processes*, 346–348.

Thomas W. Malone. 1982. Heuristics for designing enjoyable user interfaces: lessons from computer games. In: *Proc. of the 1982 Conference on Human Factors in Computing Systems*, 63–68.

Lennart Nacke. 2009. From playability to a hierarchical game usability model. In: *Proc. of the 2009 Conference on Future Play on @ GDC Canada (Future Play '09)*, 11–12.

Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In: *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, 152–158.

Janne Paavilainen. 2017. Playability: A game-centric definition. In: *Proc. of the Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '17 Extended Abstracts)*, 487–494.

David Pinelle, Nelson Wong and Tadeusz Stach. 2008. Heuristic evaluation for games: Usability principles for video game design. In: *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, 1453–1462.

Miaoqi Zhu, Xiaowen Fang, Susy S. Chan, Jacek Brzezinski. 2013. Building a dictionary of game-descriptive words to study playability. In: *Proc. of the CHI '13 Extended Abstracts on Human Factors in Computing Systems*, 1077–1082.

Miaoqi Zhu, Xiaowen Fang. 2014. Introducing a revised lexical approach to study user experience in game play by analyzing online reviews. In: *Proc. of the 2014 Conference on Interactive Entertainment*, 27:1–27:8.

Miaoqi Zhu, Fan Zhao, Xiaowen Fang and Christopher Moser. 2017. Developing playability heuristics based on nouns and adjectives from online game reviews. *International Journal of Human-Computer Interaction* 33, 3, 241–253.

Liite 1. HEP-heuristiikkakokoelma [Desurvire *et al. 2004*]

	Heuristic and Description
Game Play 1	Player's fatigue is minimized by varying activities and pacing during game play.
2	Provide consistency between the game elements and the overarching setting and story to suspend disbelief.
3	Provide clear goals, present overriding goal early as well as short-term goals throughout play.
4	There is an interesting and absorbing tutorial that mimics game play.
5	The game is enjoyable to replay.
6	Game play should be balanced with multiple ways to win.
7	Player is taught skills early that you expect the players to use later, or right before the new skill is needed.
8	Players discover the story as part of game play.
9	Even if the game cannot be modeless, it should be perceived as modeless.
10	The game is fun for the Player first, the designer second and the computer third. That is, if the non-expert player's experience isn't put first, excellent game mechanics and graphics programming triumphs are meaningless.
11	Player should not experience being penalized repetitively for the

	same failure.
12	Player's should perceive a sense of control and impact onto the game world. The game world reacts to the player and remembers their passage through it. Changes the player makes in the game world are persistent and noticeable if they back-track to where they've been before.
13	The first player action is painfully obvious and should result in immediate positive feedback.
14	The game should give rewards that immerse the player more deeply in the game by increasing their capabilities (power-up), and expanding their ability to customize.
15	Pace the game to apply pressure but not frustrate the player. Vary the difficulty level so that the player has greater challenge as they develop mastery. Easy to learn, hard to master.
16	Challenges are positive game experiences, rather than a negative experience (results in their wanting to play more, rather than quitting).
Game Story 1	Player understands the story line as a single consistent vision.
2	Player is interested in the story line. The story experience relates to their real life and grabs their interest.
3	The Player spends time thinking about possible story outcomes.
4	The Player feels as though the world is going on whether their character is there or not.
5	The Player has a sense of control over their character and is able to use tactics and strategies.
6	Player experiences fairness of outcomes.
7	The game transports the player into a level of personal involvement emotionally (e.g., scare, threat, thrill, reward, punishment) and viscerally (e.g., sounds of environment).
8	Player is interested in the characters because (1) they are like me; (2) they are interesting to me, (3) the characters develop as action occurs.
Mechanics 1	Game should react in a consistent, challenging, and exciting way to the player's actions (e.g., appropriate music with the action).
2	Make effects of the Artificial Intelligence (AI) clearly visible to the player by ensuring they are consistent with the player's rea-

	sonable expectations of the AI actor.
3	A player should always be able to identify their score/status and goal in the game.
4	Mechanics/controller actions have consistently mapped and learnable responses.
5	Shorten the learning curve by following the trends set by the gaming industry to meet user's expectations.
6	Controls should be intuitive, and mapped in a natural way; they should be customizable and default to industry standard settings.
7	Player should be given controls that are basic enough to learn quickly yet expandable for advanced options.
Usability 1	Provide immediate feedback for user actions.
2	The Player can easily turn the game off and on, and be able to save games in different states.
3	The Player experiences the user interface as consistent (in control, color, typography, and dialog design) but the game play is varied.
4	The Player should experience the menu as a part of the game.
5	Upon initially turning the game on the Player has enough information to get started to play.
6	Players should be given context sensitive help while playing so that they do not get stuck or have to rely on a manual.
7	Sounds from the game provide meaningful feedback or stir a particular emotion.
8	Players do not need to use a manual to play game.
9	The interface should be as non-intrusive to the Player as possible.
10	Make the menu layers well-organized and minimalist to the extent the menu options are intuitive.
11	Get the player involved quickly and easily with tutorials and/or progressive or adjustable difficulty levels.
12	Art should be recognizable to player, and speak to its function.

Liite 2. Internetarvioihin perustuva heuristiikkakokoelma [Zhu *et al.* 2017]

ID	Main category	Playability heuristic rule	Number of supporting reviews
CT#1	Creativity	Game players appreciate novelty and designers should try to find the right blend of novelty and familiarity	181
CT#2	Creativity	The game play elements (e.g., story, tasks) may be unpredictable so it can maintain user curiosity and interest	1004
CT#3	Creativity	Graphic shading/rendering techniques should help achieve creative goals. For instance, in racing or sports, non-realism character design, such as of cel-shaded cartoon style may attract more players	1894
CT#4	Creativity	Physics should be appropriate for the setting and mechanics. Sometimes it doesn't need to obey the laws of the real world, but it should be consistent and understandable	101
CT#5	Creativity	The game should empower the players with more creative freestyle features	296
CT#6	Creativity	Game play components should not be observed as over-used in the game	118
CT#7	Creativity	Overly familiar game elements can be perceived as novel and trendy when controlled with gestures	235
PB#1	Playability	Sports games should provide players with options for camera angles and viewpoints	1060
PB#2	Playability	Games should provide players with rich, varied statistics and analyses when appropriate. However, designers should strive to find a balance offering rich statistics and overwhelming players with information	169
PB#3	Playability	The lip movement needs to be correctly synchronized to voice-acting	465
PB#4	Playabil-	Players desire extensive customization fea-	2137

	ity	tures in many contexts. For example, the character customization modes should allow for players to define the appearance of characters as well as variety of variables	
PB#5	Playability	Designers should strive to include a variety of elements and features for players to explore and manipulate	4425
PB#6	Playability	In certain games, users expect to see realistic representations of real-life characters such as athletes	2654
PB#7	Playability	Non-player characters should be realistic and useful to facilitate game play	106
PB#8	Playability	Art should facilitate storytelling with a goal of immersing users to achieve harmonious design	1467
PB#9	Playability	Replayability is most strongly supported by an abundance of features. These also include multiplayer gaming, constant update, and the support of customization	709
PB#10	Playability	Diversifying multiplayer mode (e.g., "Eight-player") may help keep user interest thus improve replay value	238
PB#11	Playability	Multiplayer games should support various strategies and styles of play	1550
PB#12	Playability	Tension should be engendered through a combination of mechanics and environment but designers should be aware of crossing the line into frustration or burning players out	296
PB#13	Playability	Art design in game should help promote presence, immersion and player engagement	407
PB#14	Playability	A balanced gameplay should not over-power any elements (e.g., weapon) so players have difficulty in finishing the tasks	190
PB#15	Playability	Maneuverability should be balanced in order to maintain relative balance/parity or provide greater value in higher level game assets	414
PB#16	Playabil-	Artificial intelligence should seamlessly sim-	191

	ity	ulate the behavior of a human player	
PB#17	Playability	Statistics, such as those related to characters or actions, should be meaningful and communicated in understandable, satisfying, and simple ways	304
PB#18	Playability	Lifelike subtleties in character appearance (hair, skin tone, and clothing) and movement (blinking, gait) have a profound effect on perceived realism	1734
PB#19	Playability	Users desire to create and customize own contents (e.g., playlist)	350
PB#20	Playability	Team-based multiplayer game modes should emphasize cooperation through adding incentives and creative mechanics	765
PB#21	Playability	The game environment must be comfortable to view and help present the story	891
PB#22	Playability	Exaggerated, cartoon-like visuals can increase appeal to children players	106
PB#23	Playability	Side quests are highly desirable and should provide suitable challenge as well as non-trivial outcomes and rewards	551
PB#24	Playability	Level grinding should be sufficiently arduous so as to facilitate a sense of accomplishment when new levels are reached but not so tedious that players become frustrated	208
PB#25	Playability	The sound should correctly interact with the visual to immerse users in the atmosphere of the game	889
PB#26	Playability	There are multiple forms and methods (e.g., character, cut-scene) through which the game's story is presented	1860
PB#27	Playability	Any artistic elements (e.g., animation, sound) to be added should fit the game specifically	179
PB#28	Playability	Actions should result in sensible outcomes that increase the player understanding of the game mechanics and environment	684
PB#29	Playability	The game storylines should make sense to players	6569

PB#30	Playability	Pick-up-and-play features is useful for mobile and/or multiplayer party games	108
PB#31	Playability	Pass-and-play is useful for board-game style multiplayer gaming. The play experience can be augmented by controller-specific screens that can show unique content to each player during their turn	41
PB#32	Playability	A goal of multiplayer gaming should be to emphasize social interactions and discovery for its own sake	244
PB#33	Playability	Game-play mode (e.g., local multiplayer) should be functionally compatible with console modules (e.g., WIFI, camera)	92
PB#34	Playability	High-quality pre-match presentations can help engage player	51
PB#35	Playability	Virtual artifacts, such as in-game sports memorabilia, can be effective, satisfying, and rewarding	1642
PB#36	Playability	A game should have rich features such as various game modes and unique challenge/rewards	1624
PB#37	Playability	In sports games, emulating real physicality and personality of famous players may entertain the players who like/love them	2532
PB#38	Playability	Players should be able to alter the default settings (e.g., control setting) to support own play styles	3463
PB#39	Playability	Context-sensitivity can be used to make games more accessible or to reduce the steps necessary to complete repeated tasks	225
PB#40	Playability	The assigned tasks should result in well-balanced play and user activities	7602
PB#41	Playability	The A.I. of a game is perceived useful for accomplishing tasks	299
PB#42	Playability	The game should allow users to rematch and/or replay at any point	642
PB#43	Playability	Humor is highly desirable in most game and is congruent with a wide array of elements,	4815

		atmospheres, and environments	
PB#44	Playability	If the game story takes place in a real city or area, a realistic environment design in the game may gain players unsurpassed commendation	790
PB#45	Playability	Classic control scheme may still be wanted in new games of a classic category, as players may expect specific control schemes in given genres and/or subgenres	1535
PB#46	Playability	Accurate game mechanism such as collision-detection is necessary to sustain player satisfaction, especially in online multiplayer gaming	406
PB#47	Playability	Overt encouragement is desirable in motion-based exercise games	265
PB#48	Playability	Design and arrangement of intended movement should be reasonable in motion-controlled dance games	164
PB#49	Playability	Rhythm-based games should have responsive controls and meaningfully express player actions	99
PB#50	Playability	Players enjoy the games with good storyline that is moderated by multiplayer features	102
PB#51	Playability	The game A.I. should be perceived as “clever” to evoke a sense of being challenged for players of different levels	299
PB#52	Playability	Level or location-specific challenges foster exploration, satisfaction/accomplishment and thus replayability	3029
PB#53	Playability	The ease and efficiency of controlled movement can be altered by adjusting character level and equipped items, etc.	47
PB#54	Playability	The game may allow players to replay using different characters in the game	4419
PB#55	Playability	The game should carefully heightened and maintained user emotion/sensation by deliberately arranging the tasks at each level	78
PB#56	Playabil-	Players desire a high degree of freedom of	4910

	ity	movement, exploration, and tactics	
PB#57	Playability	Controls should be as easy to learn without compromising schemes for more complex gameplay	13604
PB#58	Playability	Linear game play may lead to user boredom	739
PB#59	Playability	A game should have more connectivity capability to support multiplayer-related features	1450
UB#1	Usability	The capacity for customization of game elements is generally desirable and the game may give instructions or self-design features on how to customize its gameplay	645
UB#2	Usability	Players expect reliable and consistent physics that correctly corresponds to the setting and tone	281
UB#3	Usability	Games should simplify controls and sometimes provide instant learnability	1224
UB#4	Usability	Notifications can take a variety of forms and can be expressed through a variety of narrative devices and/or mechanisms. However, they should be properly positioned on the screen so users can notice it immediately, but they should not intrude game-play	162
UB#5	Usability	Peripherals should be designed with ergonomics, safety, and reasonable levels of accessibility in mind	406
UB#6	Usability	Game control should allow a smooth gaming experience without unnecessary pauses	265
UB#7	Usability	Users can safely save their progress at any states; The availability of game saving may affect challenge/difficulty	120
UB#8	Usability	For the same game, controls must be consistently functional across multiple platforms to ensure the same playfulness	912
UB#9	Usability	Keypad design, which allows players to input data or control the game, should be user-friendly. For example, the design of the key-	115

		pad should be accommodated to hand/finger sizes of game players	
UB#10	Usability	Maps in the game should be easy to navigate, interpret, and manipulate and should correspond closely to player mental models for game environments	607
UB#11	Usability	UI elements (e.g., cursor) may be deliberately designed to provide feedback on certain game operations (e.g., user status)	112
UB#12	Usability	Game UI should reasonably respond to user actions	385
UB#13	Usability	The time spent on loading and/or saving game should be minimized, as it may impact enjoyment of game-playing	591
UB#14	Usability	Content-wise, a game should be error-free (e.g., typo)	112
UB#15	Usability	Calibrating settings and specialized controllers should be efficient and understandable	146
UB#16	Usability	Accurate, real-time feedback through avatar is highly desirable in motion-based exercise games	129
UB#17	Usability	The default camera angles should provide unobstructed views that are appropriate so not to affect game playing	150
UB#18	Usability	Users should be able to see distant objects at a stable frame-rate and the game should allow players to customize the draw distance to balance the game performance and visuals	118
UB#19	Usability	Design of game control scheme needs to provide conveniences to players	407
UB#20	Usability	Players can quickly save a game with a single input so as to not disrupt game play	112
UB#21	Usability	In-game tools and menus should be usable in the manner as any other productivity application	77
UB#22	Usability	Control systems should respond to user actions accurately especially mapped through external devices/units	266

UB#23	Usability	Players may expect auto-save features at certain points especially after cut-scenes and boss fights	197
UB#24	Usability	Control mechanisms should be consistent in different gaming events	1121

Neljäs teollinen vallankumous ja sen vaikutukset teollisuuden yrityksiin

Roni Tyrväinen

Tiivistelmä.
Tässä tutkielmassa käsitellään teollisuudessa kiihtyvään tahtiin etenevää muutosta, jota kutsutaan myös neljänneksi teolliseksi vallankumoukseksi. Teollisella vallankumouksella tarkoitetaan radikaalia muutosta teollisuudessa, johon voisi esimerkiksi sisältyä suuret muutokset teollisuuden toimintamenetelmissä tai tuotannon tehostuminen merkittävästi uusien teknologioiden hyödyntämisen myötä. Edellisten kolmen vallankumouksen tapaan myös neljäs teollinen vallankumous tulee muuttamaan teollisuutta radikaalisti. Tutkielmassani käsittelenkin neljännen teollisen vallankumouksen vaikutuksia teollisuuden yrityksiin kirjallisuuskatsauksen muodossa. Neljäs teollinen vallankumous muodostuu yhdeksästä eri osa-alueesta, jotka ovat erilaisia teknologioita tai toimintamenetelmiä. Suurin osa osa-alueista on jo osittain joidenkin yritysten käytössä, mutta samaan aikaan moni osa-alue tarvitsee myös lisää kehitystyötä ja teknologioiden käyttöönottoa, jotta neljäs teollinen vallankumous todella tapahtuisi kunnolla. Tutkielmassani avaan näiden teknologioiden ja toimintamenetelmien ominaisuuksia, käyttökohteita sekä niiden tuomia etuja yrityksille. Vaikka neljäs teollinen vallankumous tuokin monia uusia etuja yrityksille, sillä on myös omat uhkakuvansa, joita myös käsittelen tutkielmassani.

Avainsanat ja -sanonnat: Neljäs teollinen vallankumous, teollisuus 4.0, teollinen internet.

1. Johdanto

Teollisen vallankumouksen käsite yhdistetään usein vuosien 1760 ja 1840 välillä tapahtuneeseen ensimmäiseen teolliseen vallankumoukseen. Tämän jälkeen on kuitenkin ehtinyt tapahtua jo kaksi muuta teollista vallankumousta. Toinen teollinen vallankumous alkoi 1800- ja 1900-lukujen vaihteessa, kun yritykset alkoivat hyödyntää tuotannoissaan sekä massatuotantoa että sähköä. Kolmas teollinen vallankumous alkoi 1960-luvulla yritysten alkaessa hyödyntää tietotekniikkaa ja pienelektroniikkaa tuotannoissaan. Neljäs teollinen vallankumous on jo alkanut ja se tulee jatkumaan seuraavien vuosikymmenien ajan. Se rakentuu osittain kolmannen teollisen vallankumouksen päälle, muodostuen uusista, tehokkaammista mutta halvemmista teknologioista, toimintamenetelmistä ja laitteista. [Schwab 2016.]

Vaikka neljäs teollinen vallankumous onkin vasta aluillaan, se tulee kolmen edellisen teollisen vallankumouksen tapaan tuomaan suuria muutoksia mukanaan. Neljäs teollinen vallankumous perustuu pitkälti paremmaksi muuttuneisiin tiedonkeräys- ja analysointimenetelmiin sekä näiden pohjalta autonomisia päätöksiä tekeviin laitteisiin ja järjestelmiin. Tiedonkeruussa käytetään avuksi erilaisia sensoreita sekä uusissa että vanhoissa laitteissa. Sensoreista ja laitteista kerätty data välitetään muille laitteille ja järjestelmille Internetin välityksellä. Data analysoidaan automaattisesti ja tekoälyn avulla laitteet tai järjestelmät osaavat hyödyntää näitä analyysejä esimerkiksi optimoimalla toimintaansa. Tämän lisäksi koko yrityksen oma tiedonkulku paranee ja yritysten välillä tapahtuva kommunikointi tulee selkeämmäksi ja nopeammaksi. [Rüßmann *et al.* 2015.] Näiden uusien teknologioiden avulla yritykset voivat parantaa tuotantoaan tehden siitä nopeamman, tehokkaamman ja muutoksille sopeutuvamman.

Aihe on hyvin ajankohtainen ja motiivina tutkielman teolle onkin antaa tietoa siitä, mikä neljäs teollinen vallankumous ylipäätään on, mistä osa-alueista se muodostuu ja mitä muutoksia se tuo teollisuuden yrityksille. Luvussa 2 avaan neljännen teollisen vallankumouksen käsitettä ja sen muodostavia eri osa-alueita omina kohtinaan. Kohdissa avaan kunkin tarkasteltavan osa-alueen käsitettä ja kerron sen hyödyistä ja mahdollisista uhkakuvista. Luvussa 3 käsittelen neljännen teollisen vallankumouksen tuomia hyötyjä ja uhkakuvia isompana kokonaisuutena. Luku 3 jakaantuu kahteen osaan; ensimmäinen kohta kertoo konkreettisemmin neljännen teollisen vallankumouksen osa-alueiden kokonaisuudesta eli älytehtaasta, ja toinen kohta keskittyy lähinnä yrityksen taloudellisiin asioihin. Luku 4 on yhteenveto tutkielmasta. Aihetta käsitellään kirjallisuuskatsauksen muodossa ja tietoa on kerätty mahdollisimman laajasti eri tietokannoista. Koska tutkielman aihe on ajallisesti melko uusi, hyviä lähteitä erityisesti lukuun 3 oli saatavilla valitettavan rajallisesti. Tämän vuoksi luku 3 perustuukin vain muutamaan keskeiseen lähteeseen.

2. Neljännen teollisen vallankumouksen osa-alueet

Neljäs teollinen vallankumous tai lyhyesti *teollisuus 4.0* (industry 4.0) tarkoittaa jo alkanutta ja seuraavina vuosikymmeninä jatkuvaa suurta kehitystä yritysten arvoketjuissa. Termi teollisuus 4.0 on käytössä erityisesti Saksassa ja muualla Euroopassa. [Schlaepfer *et al.* 2015.] Termi onkin peräisin Saksasta, jossa valtio on pyrkinyt edistämään teollisuuden seuraavaa vallankumousta [Schwab 2016]. Saksan esimerkkiä ovat myös seuranneet monet muut maat kuten Yhdysvallat ja Japani [Schuh *et al.* 2017]. Näissä maissa on otettu käyttöön myös muita termejä kuten *teollinen internet* (industrial internet), kuvaamaan neljättä teollista vallankumousta, vaikka teollisella internetillä tarkoitetaankin muuten

hieman eri asiaa. Vaikka termien käytössä ja niiden sisällöissä on eroja, kaikkia niitä kuitenkin yhdistää se, että ne kuvaavat tulevaa radikaalia muutosta yritysten tuotanto- ja arvoketjuihin, sekä ihmisten elämiin. [Schlaepfer *et al.* 2015.]

Neljäs teollinen vallankumous muodostuu useista eri osa-alueista. Nämä osa-alueet ovat esimerkiksi uusia teknologioita tai toimintamenetelmiä. [Rüßmann *et al.* 2015.] Osa neljännen teollisen vallankumouksen muodostavista osa-alueista, kuten esimerkiksi big data -analytiikka tai pilvipalvelut ovat jo osittain käytössä joissakin yrityksissä. Moni näistä teknologioista kaipaa kuitenkin vielä lisää kehitystyötä, eikä teknologioita käytetä nykyisin kuin yksittäisinä osina yritysten arvoketjuja. Osa-alueita ei ole myöskään integroitu toisiinsa kuten neljännen teollisen vallankumouksen tarkoituksena olisi tehdä. [Rüßmann *et al.* 2015.] Toisin sanoen, vaikka neljäs teollinen vallankumous onkin joidenkin osa-alueiden käytön osalta lähtenyt käyntiin, sen suurin potentiaali on vielä saavuttamatta. Suurin potentiaali saavutetaan, kun yritykset ryhtyvät hyödyntämään kaikkia osa-alueita samanaikaisesti. Neljäs teollinen vallankumous muodostuu yhdeksästä eri osa-alueesta [Rüßmann *et al.* 2015]:

- Autonomiset robotit
- Teollinen internet
- Big data -analytiikka
- Järjestelmien täysi integraatio
- Kyberturvallisuus
- Lisätty todellisuus
- Materiaalia lisäävä valmistus
- Pilvipalvelut
- Simulaatio

2.1 Autonomiset robotit

Autonomiset robotit ovat yksi keskeisimpiä neljännen teollisen vallankumouksen muodostavia teknologioita. Robotteja on käytetty jo pitkään yritysten tuotannoissa, mutta nyt robotit ovat muuttumassa itsenäisiksi, tehokkaammiksi, yhteistyökykyisemmiksi, turvallisemmiksi ja mukautuvammiksi muutoksille. Tämän lisäksi tulevaisuuden robotit eivät aiheuta yrityksille yhtä suuria kustannuksia, vaikka ne samalla ovatkin huomattavasti nykypäivän robotteja kykeneväisempiä selviytymään erilaisista tehtävistä. [Rüßmann *et al.* 2015.]

Autonomisesti toimivat robotit kykenevät suorittamaan niille annettavat tehtävät itsenäisesti. Ne kykenevät myös suoriutumaan monista odottamattomista ongelmista ilman ihmisen apua. Robottien muuttuessa yhteistyökykyisemmiksi ja turvallisemmiksi niitä ei ole tarpeen sulkea tilaan, jossa ne ovat eristettyinä ihmisistä, jonka seurauksena ne kykenevät laajempaan yhteistyö-

hön ihmisten kanssa. Tällöin robotteja voidaan hyödyntää myös monissa uusissa tehtävissä, kuten logistiikassa. [Bahrin *et al.* 2016.] Autonominen logistiikka perustuu itseajaviin ajoneuvoihin, joita esimerkiksi Google on kehittänyt jo jonkin aikaa [Schwab 2016]. Kaikki edellä mainittu tehostaa yritysten tuotantoa ja lähentää eri prosesseja yrityksen arvoketjussa [Bahrin *et al.* 2016]. Roboteille voi siis tulevaisuudessa antaa monia tehtäviä, jotka ihmiset hoitavat tällä hetkellä. Tällaisissa tehtävissä on mahdollisesti joitain odottamattomia tilanteita, joista ei-autonomiset robotit eivät kykene selviämään, kuten esimerkiksi tavaroiden kuljettaminen tehtaissa. Autonominen robotti sen sijaan kykenee varomaan muita robotteja ja ihmisiä liikkuessaan tehtaassa.

Kun yritys alkaa hyödyntää robotteja tuotannoissaan, voi tästä seurata huonoa mainetta yritykselle. Autonomisten robottien tarkoituksena on tehostaa työn tuottavuutta ja turvallisuutta, eikä korvata ihmisiä kaikissa työtehtävissä [Bahrin *et al.* 2016; Rüßmann *et al.* 2015; Weyer *et al.* 2015]. Siitä huolimatta osa ihmisten suorittamista työtehtävistä korvataan autonomisilla roboteilla ja tämä saattaa aluksi aiheuttaa irtisanomisia, jotka voivat tuntua osalle ihmisistä eettisesti väärältä. Rüßmannin ja muiden [2015] mukaan ihmisten koulutustaso tulee korostumaan tulevaisuuden työelämässä, joten yritykset voisivat esimerkiksi uudelleen kouluttaa robottien korvaamaa työvoimaa uusiin tehtäviin irtisanomisten sijaan. Tämä hyödyttäisi molempia osapuolia.

2.2 Teollinen internet

Teollinen internet (industrial internet of things) on osa *esineiden internetiä* (internet of things). Esineiden internetillä tarkoitetaan erilaisia laitteita, jotka hyödyntävät sitä, että ne on yhdistetty johonkin järjestelmään tai palveluun Internet-protokollan avulla. [Thames and Schaefer 2016.] Thamesin ja Schaeferin [2016] mukaan teollinen internet tarkoittaa lähes samaa asiaa kuin esineiden internet. Ainoa ero on se, että teollinen internet keskittyy teollisuuteen ja näin ollen yhdistettävät laitteet ja niiden internetiin yhdistämiseen ajavat syyt ovat hieman erilaisia [Thames and Schaefer 2016]. Teolliseen internetiin yhdistettyjä laitteita voivat olla esimerkiksi erilaiset sensorit, robotit tai jopa keskeneräiset tuotteet tuotantolinjastolla. Nämä yhdistetyt laitteet voivat saada Internet-protokollan kautta käskyjä suorittaa joitain toimintoja tai kerätä dataa, jonka laitteet analysoivat itse tai lähettävät sen eteenpäin jonkin toisen laitteen tai järjestelmän analysoitavaksi [Thames and Schaefer 2016].

Teollinen internet tuo uusia ominaisuuksia ja käyttömahdollisuuksia laitteen loppukäyttäjälle, oli se sitten yksittäinen ihminen, tietojärjestelmä tai yritys. Teollisen internetin avulla pyritään lähinnä tehostamaan yrityksen toimintoja, kun taas esineiden internet kuvaa sen lisäksi myös monia muita asioita,

kuten erilaisia älylaitteita. Teollinen internet on yksi tärkeimmistä, ellei jopa tärkein neljännen teollisen vallankumouksen mahdollistavista teknologioista [Thames and Schaefer 2016]. Suurin osa neljännen teollisen vallankumouksen muodostavista teknologioista perustuu juuri siihen, että ne ovat yhteydessä muihin laitteisiin ja järjestelmiin ja keräävät näistä mahdollisimman paljon dataa. Tähän asti teollisen internetin hyötyjä ei ole päästy vielä hyödyntämään täysin, mutta suoritintehon muuttuessa jatkuvasti tehokkaammaksi ja halvemmaksi on todennäköistä, että yhä useampi teollinen laite ja järjestelmä kerää dataa ja jakaa tätä eteenpäin Internetin välityksellä [Schwab 2016].

Kerättyä dataa analysoimalla (ks. kohta 2.3) saadaan suuria hyötyjä yritykselle, sillä se mahdollistaa esimerkiksi kohdassa 2.1 esiteltyjen autonomisten robottien toiminnan. Teollisen internetin suurimpia uhkia ovat mahdolliset tietojärjestelmiin murtautumiset (ks. kohta 2.5), jotka voisivat pahimmillaan vaikuttaa koko yhteiskuntaan [Schwab 2016]. Schwabin [2016] mukaan myös ihmisten yksityisyyden suoja on ongelma esineiden internetissä. Teollisessa internetissä se lienee pienempi ongelma, sillä kerätty data tulee lähinnä teollisilta laitteilta ja koneilta, jotka eivät ole suoraan yhteydessä ihmisen yksityisasioihin [Lee *et al.* 2014].

2.3 Big data -analytiikka

Teollisen internetin (ks. kohta 2.2) myötä lähes kaikki laitteet pystyvät olemaan yhteydessä toisiinsa. Monet laitteet varustetaan erilaisilla sensoreilla, joilla voidaan mitata eri asioita. Tämän myötä on mahdollista kerätä suuria määriä dataa kaikista tuotannon vaiheista, analysoida tätä dataa ja analyysien avulla tehostaa tuotantoa eri keinoin. Big data -analytiikkaa hyödynnetään jo joissakin yrityksissä, mutta yleensä big data -analytiikka ja sen kerääminen koskevat vain ihmisten tuottamaa, tai ihmisiin liittyvää dataa. Neljännen teollisen vallankumouksen big data -analytiikka ja datan kerääminen koskevat teollisten järjestelmien ja laitteiden omaa dataa ja sen käsittelyä. [Lee *et al.* 2014.]

Big data -analytiikkaa voivat hyödyntää sekä ihmiset että erilaiset laitteet ja järjestelmät. Esimerkiksi kohdassa 2.1 käsitellyt autonomiset robotit voivat hyödyntää päätöksenteossaan järjestelmien ja laitteiden keräämää dataa, sekä sen pohjalta tehtyjä analyysejä. Romeron ja muiden [2016] mukaan myös tuotannontyöntekijät hyötyvät big data -analytiikasta. Big data -analytiikan avulla voidaan ennustaa tulevaa ja työntekijät, erityisesti tuotannonjohtajat voivat hyödyntää tätä suunnitellessaan tuotannon tulevaa toimintaa. Big data -analytiikkaa voivat hyödyntää myös tuotantolinjaston suunnittelijat, jotta linjastosta tulisi mahdollisimman tehokas. Myös tavallinen tuotannon linjasto-

työntekijä tulee jatkossa hyödyntämään enemmissä määrin big data -analytiikan avulla tehtyjä analyyseja. [Romero *et al.* 2016.]

Schwabin [2016] mukaan big data -analytiikan avulla voidaan myös nopeuttaa päätöksentekoa, saada päätöksenteosta mahdollisesti parempaa ja tuoda säästöjä. Toisaalta big data -analytiikan ongelma on luottamuksen luominen järjestelmiin, jotka tekevät analyyseja sekä näiden järjestelmien yleinen toimivuus. On myös tärkeää miettiä, kuka on vastuussa, jos data-analyysi epäonnistuu tai se aiheuttaa esimerkiksi loukkaantumisen tai kuoleman. [Schwab 2016.]

2.4 Yritysten tiedonkulun ja tietojärjestelmien täysi integraatio

Nykypäivän yritykset kärsivät täyden integraation puuttumisesta. Yritykset on jaettu erilaisiin osastoihin, yritysten tietojärjestelmiä ei ole integroitu toimimaan yhdessä, eivätkä yritykset, niiden asiakkaat tai toimittajat ole niin läheisessä kanssakäymisessä kuin ne voisivat olla. Näistä syistä yritysten toiminta on tällä hetkellä jakautunut järjestelmiin ja osastoihin, jotka ovat hyviä yksittäisissä tehtävissään ja toiminnoissaan. Ne eivät kuitenkaan yleensä toimi tehokkaasti muiden järjestelmien ja osastojen kanssa, eivätkä hyödynnä niiltä saatavaa tietoa tarpeeksi. Tämä tulee kuitenkin todennäköisesti muuttumaan neljännen teollisen vallankumouksen myötä, sillä sen tavoitteena on saada yritysten kaikki toiminnot ja osastot integroitumaan yhtenäisemmiksi ja yhteistyökykyisemmiksi. Myös tiedonkulku paranee huomattavasti yrityksen sisällä ja yritysten välillä. [Rüßmann *et al.* 2015.]

Esimerkiksi kohdassa 2.1 käsitellyt autonomiset robotit, jotka työskentelevät yhdessä ihmisten kanssa ovat hyvä esimerkki integraation lisäämisestä yrityksen tuotannossa. Nykyään robotit työskentelevät eristettyinä ihmisistä [Bahrin *et al.* 2016], jolloin tarvitaan pidempiä tuotantolinjoja ja hukataan aikaa, kun ihminen ja robotti eivät kykene hoitamaan tehtäviään samanaikaisesti. Jatkossa robottien työskennellessä ihmisten kanssa yhdessä, voidaan joissakin tapauksissa sekä ihmisen että robotin tehtävä suorittaa samanaikaisesti ja samassa paikassa.

Yrityksen tietojärjestelmiä integroitaessa ongelmaksi voi nousta se, että yrityksellä voi olla käytössä vanhoja, niin sanottuja legacy-järjestelmiä. Nämä tietojärjestelmät on usein kehitetty toimimaan vanhalla laitteistolla ja vanhoilla ohjelmointikielillä tai tekniikoilla. Vanhojen järjestelmien täysi integraatio saattaa olla jopa mahdotonta, jolloin yrityksen pitää siten vain tyytyä osittaiseen integraatioon tai sitten kehittää tietojärjestelmänsä kokonaan uusiksi. Jälkimmäinen voi olla parempi vaihtoehto, sillä neljännen teollisen vallankumouksen yhteydessä otetaan käyttöön luultavasti täysin uusia tietojärjestelmiä, jolloin voisi olla helpompaa uusia myös yrityksen muut tietojärjestelmät.

2.5 Kyberturvallisuus

Kyberturvallisuus ei sinänsä ole uusi teknologia, mutta se on silti tärkeä osa neljättä teollista vallankumousta. Neljäs teollinen vallankumous tulee yhdistämään lähes kaikki yritysten käyttämät laitteet ja koneet Internetiin tehden ne samalla alttiimmiksi erilaisille Internetin välityksellä tehtäville hyökkäyksille [Rüßmann *et al.* 2015]. Koska kyseiset Internetiin yhdistettävät laitteet ja järjestelmät ovat tärkeitä osia yritysten tuotantoa ja muita toimintoja, ne pitäisi suojata mahdollisimman hyvin hyökkäyksiä vastaan.

Teollisille järjestelmille tärkeintä on etenkin järjestelmien ja laitteiden saatavuus. Jos järjestelmät ja laitteet eivät ole käytettävissä, tuotannon teho laskee ja yritys tekee tappiota. Palvelunestohyökkäyksiltä suojautumiseksi ei kuitenkaan riitä järjestelmien alasajo hyökkäyksen ajaksi niin kuin nykyään usein tehdään, sillä tällä on myös suora vaikutus tuotannon saatavuuteen. [Ahmad-Reza *et al.* 2015.]

Neljännen teollisen vallankumouksen myötä moni yritys alkaa keräämään paljon dataa. Tämä data käsitellään ja viedään yleensä tiettyyn paikkaan, joka voi olla jonkinlainen pilvipalvelu (ks. kohta 2.8). Tämä luo omat tietoturvaongelmansa. Kaikki varastettava tieto sijaitsee yhdessä paikassa eikä hajautettuna useisiin tietojärjestelmiin. Toisaalta esimerkiksi Leen [2015] mukaan tämä ei ole ongelma, sillä kerätty data pyritään käsittelemään reaaliaikaisesti, eivätkä kerättyä dataa ymmärrä kuin sitä käsittelevät järjestelmät ja ihmiset.

Yritysten tulisi myös valvoa tarkasti, etteivät haittaohjelmat pääse vaikuttamaan niiden tuotannon laatuun. [Ahmad-Reza *et al.* 2015.] Tällaista voisi olla esimerkiksi sabotaasi, jossa yrityksen tuotetta muokataan muuttamalla sen valmistustapoja niin, että kyseinen tuote pitää vetää markkinoilta. Rüßmannin ja muiden [2015] mukaan pitkälle kehitetty identiteetin- ja pääsynhallinta koskien sekä ihmisiä että laitteita on hyvä keino estää laitteiden ja järjestelmien väärinkäyttöä. Vaikka neljäs teollinen vallankumous tuokin yrityksille suurta potentiaalia tuotantojen tehokkuuden nostamiseen, se samalla nostaa myös tietoturvauhkia suuremmiksi. Jos tietoturvauhkiin ei varauduta kunnolla, saatu hyöty tuotannolle saattaa jäädä pieneksi.

2.6 Lisätty todellisuus

Neljännen teollisen vallankumouksen myötä myös teollisuuden tuotannontyöntekijöiden työnkuva muuttuu. Työntekijät alkavat työskennellä erilaisten uusien laitteiden kanssa ja niiden avustamana. Romero ja muut [2016] kuvailevat tällaista työntekijää termillä *operaattori 4.0* (operator 4.0). Operaattori 4.0 hyödyntää työssään uusia laitteita, jotka avustavat ja helpottavat työtehtävissä

[Romero *et al.* 2016]. Yritykset eivät siis hyödy neljännestä teollisesta vallankumouksesta vain uusien koneiden ja laitteiden muodossa, vaan myös uusiin asioihin kykenevien ja tehokkaampien työntekijöiden muodossa.

Yksi merkittävimmistä teknologioista, joita operaattori 4.0 voi hyödyntää on *lisätty todellisuus* (augmented reality). Lisätty todellisuus nimensä mukaisesti lisää reaaliaikaisesti asioita näkemäämme todellisuuteen. Lisätyt asiat voivat olla joko ääntä tai grafiikkaa, joiden avulla esimerkiksi avustetaan työntekijää suorittamaan tehtävänsä. Työntekijällä voi olla päässään esimerkiksi kuulokkeet tai lasit, jotka lisäävät tämän todellisuutta. [Romero *et al.* 2016.] Lisätyn todellisuuden avulla työntekijät saavat helpommin apua, eikä tehtävän suorittamiseen välttämättä tarvita muita henkilöitä ohjeistamaan. Myös ihmisten kouluttamisesta voi tulla lisätyn todellisuuden avulla helpompaa ja tehokkaampaa, sillä työntekijä voi työskennellä saadessaan samalla ohjeita, kun nykyisin pitäisi vaihdella esimerkiksi ohjekirjan ja työskentelyn välillä [Rüßmann *et al.* 2015; Romero *et al.* 2016]. Toisaalta koulutuksen tarve joihinkin tehtäviin vähenee, sillä lisätyn todellisuuden laitteet kykenevät myös tarkkailemaan työntekijän työskentelyä ja huomauttamaan virheistä, jos niitä tapahtuu [Romero *et al.* 2016]. Vaikka koulutuksen tarve itse työtehtävään saattaa vähentyä, pitää työntekijä silti kouluttaa lisätyn todellisuuden mahdollistavien laitteiden käyttöön. Toisaalta Rüßmannin ja muiden [2015] mukaan esimerkiksi IT-taitojen tarve nousee tulevaisuuden työelämässä ja tämä lisää kouluttautumisen tarvetta.

Lisätyn todellisuuden uhkana on lisätyn todellisuuden aiheuttama keskittymishäiriö, jossa ihminen unohtaa keskittyä ympärillä tapahtuviin asioihin [Schwab 2016]. Tämän vuoksi lisättyä todellisuutta tulisi käyttää etenkin aluksi paikassa, jossa ei ole suurta mahdollisuutta vahinkoihin. Lisäksi Schwabin [2016] mukaan riskinä on mahdolliset huonot kokemukset, jotka ajavat käyttäjät pois lisätyn todellisuuden hyödyistä. Toisaalta yhtä lailla uhkana voi olla lisätyn todellisuuden aiheuttama addiktio joka syntyy, kun lisätty todellisuus koetaan oikeaa todellisuutta paremmaksi [Schwab 2016]. Tämä tosin ei liene olevan kovin iso riski teollisuudessa käytössä olevissa lisätyn todellisuuden laitteissa, jotka on suunniteltu lähinnä työtehtävissä avustamiseen.

2.7 Materiaalia lisäävä valmistus

Materiaalia lisäävä valmistus (additive manufactoring) [Metsta 2016], kuten esimerkiksi 3D-tulostus, ei ole vielä kovin suuressa suosiossa yritysten tuotannoissa [Rüßmann *et al.* 2015]. 3D-tulostuksella tarkoitetaan prosessia, jossa valmistetaan tuote digitaalisen 3D-mallin perusteella kerros kerrokselta [Schwab 2016]. Teollisuudessa sitä käytetään tällä hetkellä lähinnä prototyyppien ja yk-

sittäisten osien valmistukseen [Rüßmann *et al.* 2015]. Tämä tosin voi muuttua, sillä Rüßmannin ja muiden [2015] mukaan simuloinnin (ks. kohta 2.9) lisääntyessä suunnittelutyössä fyysiset prototyypit vähenevät selkeästi. Materiaalia lisäävää valmistusta voidaan käyttää esimerkiksi pienien, mutta monimutkaisten tuotteiden valmistukseen [Stock and Seliger 2016]. Koska materiaalia lisäävään valmistukseen käytettävät laitteet ovat myös alkaneet muuttua halvemmiksi, tarkemmiksi ja nopeammiksi [Stock and Seliger 2016], niiden suosio osana yritysten tuotantoa noussee lähitulevaisuudessa. Kun materiaalia lisäävän valmistuksen nopeus ja tehokkuus kasvaa, sitä voidaan hyödyntää esimerkiksi kuljetusmatkojen pienentämiseen tuottajan ja asiakkaan välillä, sillä tuotantoa voidaan siirtää tai rakentaa helposti mihin tahansa. Toinen mahdollinen käyttötarkoitus on valmistaa kustannustehokkaasti yksittäisiä tai pieniä eriä asiakkaan käyttötarpeisiin räätälöityjä tuotteita. [Rüßmann *et al.* 2015.]

Tällä hetkellä suurimmat ongelmat, miksi materiaalia lisäävää valmistusta ei hyödynnetä teollisuudessa, ovat sen valmistuksen hitaus, hinta ja valmistettavien tuotteiden kokorajoitukset [Schwab 2016]. Schwabin [2016] mukaan nämä ongelmat tullaan kuitenkin ylittämään tulevaisuudessa. 3D-tulostetut tuotteet ovat myös jossain määrin heikompia niiden valmistusprosessin takia. Valmistuksessa syntyy myös normaalia tehdastuotantoa enemmän jätettä, joka rasittaa luontoa. Yrityksille on myös uhkana se, että 3D-tulostus lisääntyy merkittävästi asiakkaiden keskuudessa. Tällöin uhaksi tulee tuotteiden piratismi, brändin arvon lasku ja se, että asiakkaat eivät enää koe tarvetta ostaa tuotetta vaan ennemmin tulostavat sen. [Schwab 2016.] Toisaalta tämä mahdollistaa liiketoimintamallit, joissa keskitytään vain tuotteiden kehitykseen ja tuotteen valmistus jätetään asiakkaalle.

2.8 Pilvipalvelut

Monissa yrityksissä on jo käytössä joitakin pilvipalveluita, mutta ne eivät ole vielä levinneet laajasti teollisuudessa käytettäviin laitteisiin ja järjestelmiin [Rüßmann *et al.* 2015]. Moni neljännen teollisen vallankumouksen muodostama teknologia tarvitsee palvelun, jossa säilöä niiden keräämää ja käsittelemää dataa. Datan pitäisi myös olla saatavilla muiden järjestelmien käytettäväksi koska tahansa, sillä järjestelmien ja laitteiden täysi integrointi perustuu tiedon jakamiseen eri järjestelmien ja yritysten osastojen välillä (ks. kohta 2.4).

Lähivuosina pilvipalveluiden hinnat tulevat laskemaan merkittävästi [Schwab 2016]. Tämän lisäksi pilvipalveluiden suorituskyky ja reaktionopeus muuttuvat paljon paremmiksi. Nämä syyt johtavatkin siihen, että pilvipalveluita hyödynnetään jatkossa enemmän teollisuudessa ja yhä useampi laite ja järjestelmä on yhteydessä jonkinlaiseen pilvipalveluun. [Rüßmann *et al.* 2015.] Tha-

mes ja Schaefer [2016] kuvaavat tällaista järjestelyä pilvipohjaiseksi tuotannoksi. Pilvipohjaisen tuotannon etuja ovat sen skaalautuvuus, ketteryys ja tuotantokustannusten väheneminen samanlaisten palveluiden tullessa yhden palvelun alaiseksi [Thames and Schaefer 2016].

Pilvipalveluiden ongelmana on niiden tietoturva- ja yksityisyysongelmat. Pilvipalvelu kerää kaiken käsiteltävän tiedon yhteen palveluun ja tietoturvan pettäessä kaikki tieto saadaan varastettua kerralla. Tämä ei kuitenkaan liene kovin suuri ongelma teollisuuden pilvipalveluissa, kuten kohdassa 2.5 kävi ilmi. Yritysten tulisi silti huomioida tämä pilvipalveluihin panostaessaan.

2.9 Simulaatio

Simulaatioita voidaan käyttää erilaisten toimintamallien ja tuotteiden mallintamiseen. Rüßmann ja muut [2015] toteavat simuloimisen olevan jo käytössä esimerkiksi tuotteiden tai niiden valmistusprosessien mallintamissa, mutta simuloimista ei kuitenkaan vielä käytetä isompien asioiden simuloimiseen, kuten koko tuotantolaitoksen toiminnan kuvaamiseen. Tulevaisuudessa simulaatiota hyödynnetään erilaisten tuotantojärjestelmien ja laitteiden toiminnan simuloimisessa. Tämä on keskeinen osa neljättä teollista vallankumousta ja se mahdollistaa niin sanotut kyberfyysiset järjestelmät. [Weyer *et al.* 2015.]

Tämän lisäksi simuloinnin muuttuessa modulaarisemmaksi, tuotteiden muuttamisesta tulee nopeampaa ja helpompaa. Uudenlainen simulointi mahdollistaa nopeammat muutokset tuotannossa olevissa tuotteissa ja suunnitelmista toteutuksiin siirtymisessä. [Brettel *et al.* 2014.] Simulaatioissa voitaisiin myös hyödyntää esimerkiksi lisättyä todellisuutta (ks. kohta 2.6), jolloin simuloitavat tuotteet voisivat olla niitä suunniteltaessa oikean kokoisia alusta alkaen. Tämä myös on yhteydessä siihen, miten simulaatio vähentää fyysisten prototyyppien tarvetta merkittävästi tulevaisuudessa [Rüßmann *et al.* 2015].

3. Neljännen teollisen vallankumouksen vaikutukset yrityksiin

Neljännen teollisen vallankumouksen keskiössä ovat älykäs tuote, tuotteen kasaava älykäs kone, sekä lisättyä todellisuutta hyödyntävä tuotannontyöntekijä [Weyer *et al.* 2015]. Lisättyä todellisuutta hyödyntäviä tuotannontyöntekijöitä on jo käsitelty jonkin verran kohdassa 2.6. Weyerin ja muiden [2015] mukaan neljäs teollinen vallankumous ei pyri korvaamaan kaikkia työntekijöitä roboteilla, vaikka se perustuukin pitkälti automaation lisäämiseen yritysten tuotannoissa. Osa työntekijöistä toki korvataan roboteilla, mutta yritykset luovat samaan aikaan myös useita uusia työtehtäviä. Rüßmannin ja muiden [2015] mukaan neljäs teollinen vallankumous ennemmin jopa lisää työpaikkojen määrää, kuin vähentää sitä.

Weyerin ja muiden [2015] mukaan älykkäällä tuotteella tarkoitetaan tuotetta, joka ei ole vain kokoamistaan odottava tavallinen puoliksi valmis tuote, vaan tärkeä ja itsenäinen osa tuotantojärjestelmää. Älykäs tuote voi säilöä muistissaan esimerkiksi siihen tehtyjä muutoksia ja pitää kirjaa siitä, mitä muutoksia tuote vielä tarvitsee ennen kuin se on valmis [Weyer *et al.* 2015]. Tuotteen kokoaminen voi tämän myötä tulla täysin automatisoiduksi, kun tuotetta kokoava kone tai järjestelmä kykenee lukemaan tuotteen tarvitsemat muutokset valmistettavasta tuotteesta.

Älykkäällä koneella tarkoitetaan konetta, joka hyödyntää ainakin osaa luvussa 2 esitellyistä teknologioista, jolloin koneesta tulee niin sanottu *kyberfyysinen järjestelmä* (cyber physical system). Nämä kyberfyysiset järjestelmät voivat muodostaa yhdessä *kyberfyysisiä tuotantojärjestelmiä* (cyber physical production system). Kun yritys alkaa muuttamaan tuotantoaan ja ottaa käyttöön kyberfyysisiä tuotantojärjestelmiä, älykkäitä tuotteita ja lisättyä todellisuutta hyödyntäviä tuotannon työntekijöitä, muodostuu yrityksen tehtaasta niin kutsuttu *älytehdas* (smart factory). [Weyer *et al.* 2015.]

3.1 Älytehtaat

Älytehdas -termillä tarkoitetaan yrityksen tehdasta tai tuotantoyksikköä, jossa on ryhdytty hyödyntämään neljännen teollisen vallankumouksen eri osa-alueita ja niiden avulla muodostettuja kyberfyysisiä tuotantojärjestelmiä. Tämä tuo suuria etuja koko tuotantoteollisuudelle. Näitä etuja ovat esimerkiksi ennakoiva huolto ja tuotteiden massaräätälöinti. [Lee 2015.]

Älytehtaan toiminta perustuu pitkälti siihen, että lähes kaikkea saatavilla olevaa tietoa ryhdytään hyödyntämään tuotannon parantamiseksi. Tämä onkin kyberfyysisten laitteiden toimintatapa. Kyberfyysisten järjestelmien nimi viittaa siihen, miten laitteet ovat olemassa sekä fyysisessä että digitaalisessa maailmassa. Jokaisesta fyysisestä laitteesta on olemassa kopio digitaalisena versiona. Laitteen sensorien keräämä data muodostaa laitteen digitaalisen kopion ja muutokset datassa otetaan huomioon laitteen kopiota ja kopion seuraavia muutoksia muodostaessa. [Lee 2015.] Kyberfyysiset järjestelmät ovat siis osittain simulaatioita (ks. kohta 2.9) hyödyntäviä järjestelmiä. Ne simuloivat saamansa datan perusteella laitteen tilaa ja toimintoja. Näitä simulaatioita voi myös hyödyntää järjestelmien ohjaukseen, jos niiden kautta on mahdollista antaa järjestelmille komentoja. Kun nämä simulaatiot yhdistetään kohdassa 2.8 käsiteltyihin pilvipalveluihin, voidaan luoda järjestelmä, joka mahdollistaa usean tehtaan hallitsemisen ja valvonnan yhdestä paikasta.

Älytehtaalla käytössä olevat kyberfyysiset tuotantojärjestelmät mahdollistavat yhden älytehtaan tärkeimmistä ominaisuuksista. Tämä ominaisuus on

laitteiden ja järjestelmien *itsetietoisuus* (self-aware). Itsetietoisuudella ei tässä kontekstissa kuitenkaan tarkoiteta älyllistä itsetietoisuutta, joka esimerkiksi ihmisillä on, vaan ennemminkin järjestelmän oman tilan ja toimintojen tiedostamista ja näiden hyödyntämistä eri asioissa mahdollisimman tehokkaasti. Itsetietoinen järjestelmä kykenee tarkkailemaan järjestelmän nykyistä tilaa ja toimintoja, mutta sillä on hallussaan myös dataa järjestelmän aikaisemmista tapahtumista. Itsetietoinen järjestelmä voi vertailla nykyistä tilaa aiempiin tietoihin ja vertailun pohjalta se voi optimoida suoritustaan. Itsetietoinen järjestelmä varautuu myös sille annettuihin komentoihin ottamalla ensin huomioon oman tilansa. [Lee *et al.* 2014.] Kun järjestelmä osaa huomioida omaa tilaansa, se ei esimerkiksi yritä suorittaa toimintoa, joka voisi ylikuumentaa jonkin osan tai muuten aiheuttaa vahinkoa itselleen. Järjestelmän itsetietoisuus ja tilan arviointi tapahtuvat kohdassa 2.3 esitellyn big data -analytiikan avulla [Lee *et al.* 2014].

Itsetietoinen järjestelmä kykenee tarkkailemaan omaa tilaansa saamansa datan avulla ja sen pohjalta se voi esimerkiksi arvioida mikä on järjestelmän toimivuustila ja onko siinä havaittavissa riskiä hajoamiselle. Järjestelmää, joka kykenee arvioimaan omaa hajoamisriskiään, kutsutaan *itsehuoltavaksi* (self-maintained). [Lee *et al.* 2014.] Itsehuoltavat järjestelmät kykenevät myös analysoimaan omia toimintahäiriöisiä osia. Tämän lisäksi itsehuoltava järjestelmä kykenee arvioimaan ja välttämään mahdollisia ongelmia itsensä lisäksi myös valmistettavassa tuotteessa. [Lee 2015.] Itsehuollettavat järjestelmät kykenevät varautumaan esimerkiksi kuluvien osien hajoamiseen tilaamalla huollon jo ennen osan hajoamista täysin. Tällöin osan hajoaminen ei pääse tapahtumaan yllätyksenä ja siitä ei aiheudu yhtä suurta tuotannon keskeytystä. Uudet kyberfyysiset tuotantojärjestelmät pyritään myös rakentaa niin, että järjestelmän eri laitteiden ja osien vaihto on mahdollisimman helppoa ja nopeaa, eikä se aiheuta ongelmia järjestelmän muille osille [Weyer *et al.* 2015]. On siis mahdollista, että tuotantoa ei tarvitsisi keskeyttää lainkaan, kun järjestelmän kuluvat osat ovat helposti vaihdettavissa ja tieto vaihdontarpeesta tulee tarpeeksi ajoissa. Lee ja muut [2014] kuitenkin totesivat vielä muutama vuosi sitten, että tällaiset järjestelmät eivät vielä olleet mahdollisia lähitulevaisuudessa, sillä sen aikaiset toteutukset soveltuivat vain tietyille järjestelmille ja laitteille, eivätkä ne olleet mukautuvia uusiin sovellutuksiin.

Itsetietoiset järjestelmät ja laitteet jakavat keräämäänsä dataa myös muiden samanlaisten laitteiden tai samaa tuotetta työstävien laitteiden käytettäväksi. Itsetietoiset järjestelmät voivat hyödyntää muilta laitteilta ja järjestelmiltä saamaansa dataa oman toimintansa optimoimiseen. [Lee 2015.] Jos esimerkiksi jokin osa tuotantojärjestelmää hidastuu tai pysähtyy, tuotantojärjestelmän muut osat hidastavat tai pysäyttävät toimintonsa, kunnes ongelma saadaan ratkais-

tua. Leen [2015] mukaan itsetietoiset järjestelmät voivat myös hyödyntää muilta saamaansa dataa jopa yksittäisen tuotteen valmistuksen optimoimiseksi. Tämä vähentää tuotannon kustannuksia ja laitteiden turhaa käyttöä [Lee 2015].

Yksi merkittävimpiä muutoksia, joita kyberfyysiset tuotantojärjestelmät tuovat yritykselle on tuotteiden massaräätälöinti ja tuotannon muuttuminen ketterämmäksi. Massaräätälöinnissä jokainen tuote voidaan valmistaa hieman erilaiseksi tuotannon tehokkuuden pysyessä silti samana [Weyer *et al.* 2015]. Weyerin ja muiden [2015] mukaan tuotantolinjastoista tulee huomattavasti modulaarisempia ja joustavampia. Muuttuvat tuotantolinjastot mahdollistavat helpommin esimerkiksi tuotteeseen tehtäviä muutoksia tuotteen valmistuksen jo alettua [Weyer *et al.* 2015]. Myös viime hetken muutokset ovat helpompia toteuttaa tuotannon aikana [Lee 2015]. Asiakkaat voivat siis alkaa vaatia tuotteidensa olevan yhä enemmän juuri asiakkaan käyttötarkoituksiin sopivia. Massaräätälöinnin onnistuessa asiakastyytyväisyys varmasti lisääntyy, mutta aiheuttaako tämä pitkällä aikavälillä laatuvaatimusten nousun?

3.2 Yritysten taloudelliset muutokset

Toteutuessaan onnistuneesti, neljäs teollinen vallankumous tulee nostamaan teollisuuden tuottavuutta. Tuottavuuden nousu on tärkein tekijä parempien elinolosuhteiden ja taloudellisen kasvun luomisessa. Tuottavuuden merkittävä nousu yhdistääkin kaikkia tähän mennessä tapahtuneita teollisia vallankumouksia. [Schwab 2016.] Tuottavuuden mahdollinen nousu lähitulevaisuudessa johtuu pitkälti luvussa 2 esiteltyjen teknologioiden ja toimintamenetelmien käyttöönotosta teollisuuden yrityksissä.

Vaikka tuottavuuden nousu yleensä aiheuttaakin talouskasvua, osa ekonomeista kiistelee siitä, tuleeko neljäs teollinen vallankumous todellisuudessa aiheuttamaan merkittävää talouskasvua. Osa ekonomeista argumentoi, että voimakkain digitalisaation aiheuttama talouskasvu on jo tapahtunut. [Schwab 2016.] Toisaalta esimerkiksi Rüßmannin ja muiden [2015] mukaan teollisuuden voitot tulevat nousemaan neljännen teollisen vallankumouksen myötä Saksassa 30 miljardia euroa joka vuosi seuraavan 5-10 vuoden ajan. Tämä tarkoittaisi Saksassa vuosittain noin 1% kasvua BKT:hen [Rüßmann *et al.* 2015].

Tuottavuuden kasvu voi olla myös ratkaisu väestön ikääntymiseen. Nykyään ihmisten syntyvyys on laskussa ja tämä tulee aiheuttamaan suuria ongelmia tulevaisuudessa. Työmarkkinoilla on yhä vähemmän työikäisiä, yrittäminen ja suuria rahallisia investointeja vaativat ostokset vähenevät. Tämä on mahdollista estää lisäämällä työn tuottavuutta. Neljäs teollinen vallankumous korvaa osan tehtävistä autonomisilla roboteilla ja ihmistyöntekijöistä tulee samalla tehokkaampia. Tämä nostaa työntehokkuutta mahdollisesti sille tasolle,

että väestön ikääntyminen ei aiheuta suuria ongelmia yrityksille tai valtioille. [Schwab 2016.] On siis mahdollista, että neljännen teollisen vallankumouksen takia ei esimerkiksi tarvitse tehdä suuria korotuksia eläkeikään, eikä yritysten tarvitse pelätä työvoimapulaa.

Neljännen teollisen vallankumouksen hyödyt eivät tule automaattisesti yrityksille, vaan ne vaativat suuria investointeja ja panostusta eri asioihin teollisen vallankumouksen edistämiseksi [Schwab 2016]. Rüßmannin ja muiden [2015] mukaan neljännen teollisen vallankumouksen eri osa-alueiden käyttöönotto tulee maksamaan Saksan teollisuuden yrityksille yhteensä 250 miljardia euroa seuraavan 10 vuoden aikana. Tämän lisäksi uudet teknologiat ja toimintamenetelmät vaativat toimiakseen hyvän infrastruktuurin ja korkeakoulutettua työvoimaa. Suurin osa neljännen teollisen vallankumouksen teknologioista hyötyy nopeasta Internet-yhteydestä. [Rüßmann *et al.* 2015.] Tästä syystä olisikin hyvä, jos yritykset voisivat hyödyntää esimerkiksi nopeita valokuitu- tai 5G-yhteyksiä. Rüßmannin ja muiden [2015] mukaan myös ihmisten työssä tarvittavat taidot tulevat muuttumaan neljännen teollisen vallankumouksen myötä. Tulevaisuudessa erityisesti IT-taidot ja innovaatiokyvykkyys tulevat olemaan tärkeitä ominaisuuksia työelämässä. Yritykset vaihtavat osan työntekijöistään koneisiin, mutta toisaalta tarve korkeakoulutetulle työvoimalle, kuten kone- ja tietotekniikan asiantuntijoille kasvaa. [Rüßmann *et al.* 2015.] Myös yritysten tuotannoissa tarvitaan lisää työntekijöitä, esimerkiksi tuotannon valvonnan tehtäviin [Weyer *et al.* 2015]. Rüßmann ja muut [2015] painottavat, että sekä infrastruktuurin rakentamisessa että ihmisten koulutuksessa yritysten ja valtioiden tulisi tehdä tiiviimpää yhteistyötä parhaiden tulosten aikaansaamiseksi molemmille osapuolille.

Monet asiat tulevat muuttumaan neljännen teollisen vallankumouksen myötä ja tämä tuo omat haasteensa yrityksille. Kilpailu yritysten välillä tulee kovenemaan entisestään ja yrityksen kovin kilpailija saattaa olla nopeasti ja yllättäen menestykseen nouseva uusi yritys vanhan kilpailijan sijaan. [Schwab 2016.] Tällaiseen markkinoita mullistavaan kilpailijaan voi olla hyvin vaikea varautua. Schwabin [2016] mukaan paras tapa varautua uusiin yllättäviin kilpailijoihin on johdon kyky sopeutua nopeisiin muutoksiin, jatkuva oppiminen ja omien toimintatapojen heikkouksien tunnistaminen. Toisaalta suuria kilpailijoita ei voi olla ottamatta huomioon, sillä suuret yritykset hyötyvät edelleen koostaan, jos ne hyödyntävät sitä oikealla tavalla. Yrityksen tulisi panostaa innovaatioon ja olla hyväksyvämpi epäonnistuvia projekteja kohtaan. Hitaasti muuttuvat yritykset tulevat jäämään muista jälkeen, sillä yritysten välinen kilpailu tulee vain kovenemaan tulevaisuudessa. [Schwab 2016.]

Toinen haaste yrityksen johdolle on oikean strategian valitseminen. Sekä teollisuuden tuotantoyritykset että näiden yritysten laitteiden ja järjestelmien toimittajat joutuvat miettimään mihin neljännen teollisen vallankumouksen osa-alueisiin panostetaan ja millä tavoin näitä osa-alueita tullaan hyödyntämään. [Rüßmann *et al.* 2015.] Väärään osa-alueeseen panostaminen voi olla kohtalokasta yritykselle. Toisaalta neljäs teollinen vallankumous luo myös täysin uusia liiketoimintamalleja. Näissä uusissa liiketoimintamalleissa yhdistyvät asiakaslähtöisyys, tietotekniikan ja kerätyn datan hyödyntäminen. Toisaalta niissä yhdistyvät myös monet muut alat, jotka hyötyvät neljännen teollisen vallankumouksen eri osa-alueista. Näihin uusiin liiketoimintamalleihin keskittyminen voisi olla yritykselle hyvä strategia, vaikka onkin vaikeaa tietää, mikä liiketoimintamalli todellisuudessa tulee toimimaan tulevaisuudessa. [Schwab 2016.]

Neljäs teollinen vallankumous tuo myös paljon häiriöitä teollisuuteen, sillä moni asia yritysten tuotantoketjuissa kokee suuren muutoksen. Tuottajien logistiikka automatisoituu ja skaalautuu automaattisesti kysynnän mukaan, massaräätälöinti ja pienet tuotantomäärät tulevat taloudellisesti kannattaviksi ja tuotteiden valmistus muuttuu yhä tarkemmaksi ja virheet vähenevät huomattavasti. IT-järjestelmiä yhdistetään ja niiden välistä tiedonvaihtoa parannetaan, myös eri yritysten välillä. [Rüßmann *et al.* 2015.] Tämä syventää yritysten välistä yhteistyötä, mikä onkin Schwabin [2016] mukaan tärkeää tulevaisuudessa. Jaettaessa kerättävää tietoa ja taitoa yritykset voivat luoda jopa täysin uudenlaisia liiketoimintamalleja. Tämä kuitenkin vaatii suuria panostuksia molemmilta osapuolilta menestyvän yhteistyön löytämiseksi. [Schwab 2016.]

Neljännen teollisen vallankumouksen myötä myös asiakkaiden vaatimukset kohoavat. Jo tänäkin päivänä monet yritykset ovat asiakaslähtöisiä, mutta asiakkaiden vaatimukset tulevat kohoamaan entisestään. Esimerkiksi asiakkaiden hintatietoisuus tulee lisääntymään erilaisten hinnanvertailupalveluiden takia. [Schwab 2016.] Toisaalta myös asiakkaiden vaatimukset laatua kohtaan voivat nousta. Kohdassa 2 käsitellyn big data -analytiikan avulla tuotteen valmistuksessa tapahtuvat virheet vähenevät, mutta jos jokin virheellinen erä pääsee markkinoille, voi tästä koitua paljon huonoa mainetta yritykselle, jopa enemmän kuin nykyään. Kun dataa ryhdytään keräämään lähes kaikkialta ja monet asiat yhdistetään Internetiin, asiakkaat saattavat alkaa enemmissä määrin välittää myös omasta tieto- ja yksityisyydensuojastaan. Yritysten tulisi huomioida myös tämä strategioissaan.

4. Yhteenveto

Neljäs teollinen vallankumous tuo monia etuja teollisuuden yrityksille ja näiden yritysten asiakkaille. Vaikka etuja onkin useita, on neljännessä teollisessa vallankumouksessa myös monia ongelmia ja uhkakuvia yrityksille. Neljäs teollinen vallankumous on käsitteenä erittäin laaja ja tästä syystä joitakin etuja ja haittoja voi olla vaikea ymmärtää. Uskon, että jaettaessa neljäs teollinen vallankumous eri osa-alueisiin ja käsittelemällä näiden hyötyjä ja haittoja yksinään, voi saada paremman kuvan jokaisesta osa-alueesta. Tutkielmassa on käsitelty myös neljännen teollisen vallankumouksen muodostamien osa-alueiden kokonaisuuksia, näiden hyötyjä ja haittoja sekä neljännen teollisen vallankumouksen tuomia taloudellisia vaikutuksia yrityksille. Yrityksen ottaessa älytehtaita käyttöön yritys saa itselleen tehokkaamman, tarkemman ja halvemman tuotannon. Moni osa yrityksen toiminnassa automatisoituu ja kustannukset laskevat. Vaikka neljäs teollinen vallankumous tuokin paljon säästöjä yrityksille pitkällä aikavälillä, se vaatii myös paljon investointeja lyhyellä aikavälillä.

Tutkielmassa ei ole huomioitu kovin paljoa yhteiskunnallisia ja yksittäiseen työntekijään kohdistuvia vaikutuksia. Näitä asioita olisi hyvä tutkia lisää. Lisäksi jokainen neljännen teollisen vallankumouksen osa-alue tarvitsee lisää kehitystä, jotta neljäs teollinen vallankumous lähtisi kunnolla käyntiin. Olisi tärkeää tutkia erityisesti yrityksen osastojen ja tietojärjestelmien integroimista. Suurin hyöty neljännestä teollisesta vallankumouksesta saavutetaan, kun informaatio kulkee moitteetta osastojen ja järjestelmien välillä. On mahdollista, että suurimpia hyötyjä ei saavuteta, jos yritys yrittää rakentaa tämän kaiken vanhojen tietojärjestelmiensä päälle.

Viiteluettelo

Sadeghi Ahmad-Reza, Christian Wachsmann, and Michael Waidner. 2015. Security and privacy challenges in industrial internet of things. In: *Proc. of the 52nd Design Automation Conference (DAC),* 1-6.

Mohd Bahrin, Mohd Othman, Nor Azli, and Muhamad Talib. 2016. Industry 4.0: A review on industrial automation and robotic. *Jurnal Teknologi* 78, 6-13, 137-143.

Malte Brettel, Niklas Friederichsen, Michael Keller, and Marius Rosenberg. 2014. How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 perspective. *International Journal of Mechanical, Industrial Science and Engineering* 8, 1, 37-44.

Jay Lee. 2015. Smart factory systems. *Informatik-Spektrum* 38, 3, 230-235.

Jay Lee, Kao Hung-An, and Yang Shanhu. 2014. Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia Cirp* 16, 3-8.

Metsta. 2016. Standardisointiin osallistuminen kannattaa - Case: Lisäävä valmistus. http://www.metsta.fi/?we_objectID=33394. Katsottu 21.11.2018.

David Romero, Johan Stahre, Thorsten Wuest, Ovidiu Noran, Peter Bernus, Åsa Fast-Berglund, and Dominic Gorecky. 2016. Towards an operator 4.0 typology: a human-centric perspective on the fourth industrial revolution technologies. In: *Proc. of the International Conference on Computers & Industrial Engineering,* CIE46, 1-11.

Michael Rüßmann, Markus Lorenz, Philipp Gerbert, Manuela Waldner, Jan Justus, Pascal Engel, and Michael Harnisch. 2015. Industry 4.0: The future of productivity and growth in manufacturing industries. Boston Consulting Group, 09 April 2015. https://www.bcg.com/publications/2015/engineered_products_project_business-
ness_industry_4_future_productivity_growth_manufacturing_industries.aspx. Checked 3.10.2018.

Ralf Schlaepfer, Markus Koch, and Philipp Merkhofer. 2015. Industry 4.0 Challenges and Solutions for the Digital Transformation and Use of Exponential Technologies. https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry- 4-0-24102014.pdf. Checked 3.11.2018.

Günther Schuh, Reiner Anderl, Jürgen Gausemeier, Michael ten Hompel, and Wolfgang Wahlster. 2017. *Industrie 4.0 Maturity Index. Managing the Digital Transformation of Companies (acatech STUDY).* Herbert Utz Verlag, Munich.

Klaus Schwab. 2016. *The Fourth Industrial Revolution.* World Economic Forum.

Tim Stock and Günther Seliger. 2016. Opportunities of sustainable manufacturing in industry 4.0. *Procedia Cirp* 40, 536-541.

Lane Thames and Dirk Schaefer. 2016. Software-defined cloud manufacturing for industry 4.0. *Procedia Cirp* 52, 12-17.

Stephan Weyer, Mathias Schmitt, Moritz Ohmer, and Dominic Gorecky. 2015. Towards Industry 4.0-Standardization as the crucial challenge for highly modular, multi-vendor production systems. *Ifac-Papersonlin* 48, 3, 579-584.